

CRADLE: EMPOWERING FOUNDATION AGENTS TOWARDS GENERAL COMPUTER CONTROL

Anonymous authors
 Paper under double-blind review



Figure 1: The **CRADLE** framework empowers nascent foundation models to perform complex computer tasks via the same unified interface humans use, *i.e.*, screenshots as input and keyboard & mouse operations as output.

ABSTRACT

Despite the success in specific scenarios, existing foundation agents still struggle to generalize across various virtual scenarios, mainly due to the dramatically different encapsulations of environments with manually designed observation and action spaces. To handle this issue, we propose the **General Computer Control (GCC)** setting to restrict foundation agents to interact with software through the most unified and standardized interface, *i.e.*, using screenshots as input and keyboard and mouse actions as output. We introduce **CRADLE**, a modular and flexible LMM-powered framework, as a preliminary attempt towards GCC. Enhanced by six key modules: Information Gathering, Self-Reflection, Task Inference, Skill Curation, Action Planning, and Memory, **CRADLE** is able to understand input screenshots and output executable code for low-level keyboard and mouse control after high-level planning, so that **CRADLE** can interact with any software and complete long-horizon complex tasks without relying on any built-in APIs. Experimental results show that **CRADLE** exhibits remarkable generalizability and impressive performance across four previously unexplored commercial video games, five software applications, and a comprehensive benchmark, OS-World. To our best knowledge, **CRADLE** is the first to enable foundation agents to follow the main storyline and complete one-hour-long real missions in the complex AAA game Red Dead Redemption 2 (RDR2). **CRADLE** can also create a city with nearly a thousand people in Cities: Skylines, farm and harvest parsnips in Stardew Valley, and trade and bargain with a maximum weekly total profit of 87% in Dealer’s Life 2. **CRADLE** can not only operate daily software, like Chrome, Outlook, and Feishu, but also edit images and videos using Meitu and CapCut. With a unified interface to interact with any software, **CRADLE** greatly extends the reach of foundation agents by enabling the easy conversion of any software, especially complex games, into benchmarks to evaluate agents’ various abilities and facilitate further data collection, thus paving the way for generalist agents. Video demos and code can be found at <https://cradle2024acc.github.io/Cradle>.

1 INTRODUCTION

Artificial General Intelligence (AGI) has long been a north-star goal for the AI community (Morris et al., 2023). The recent success of foundation agents, *i.e.*, agents empowered by large multimodal models (LMMs) and advanced tools, in various environments, *e.g.*, web browsing (Zhou et al., 2023; Deng et al., 2023; Gur et al., 2023; Zheng et al., 2024b;a; He et al., 2024), operating mobile applications (Yang et al., 2023b; Wang et al., 2024b) and desktop software (Zhang et al., 2024; Wu et al., 2024), crafting and exploration in Minecraft (Wang et al., 2023b; 2024a; 2023a), and some robotics scenarios (Huang et al., 2022; Brohan et al., 2023b; Driess et al., 2023; Brohan

et al., 2023a), have shown promise. However, current foundation agents still struggle to generalize across different scenarios, primarily due to the dramatic differences in the encapsulation of environments with human-designed observation and action space. Therefore, developing foundation agents applicable to various environments remains extremely challenging.

Computers, as the most important and universal interface that connects humans and the increasing digital world, provide countless rich software, including applications and realistic video games for agents to interact with, while avoiding the challenges of robots in reality, such as hardware requirements, constraints of practicability, and possible catastrophic failures (Raad et al., 2024). Mastering these virtual environments is a promising path for foundation agents to achieve generalizability. Therefore, we propose the **General Computer Control** (GCC) setting¹:

Building foundation agents that can master ANY computer task via the universal human-style interface by receiving input from screens and audio and outputting keyboard and mouse actions.

There are many challenges to achieving GCC: i) good alignment across multi-modalities for better understanding and decision-making; ii) precise control of keyboard and mouse to interact with the computer, which has a large hybrid action space, including not only which key to press and where the mouse to move, but also the duration of the press and the speed of the mouse movement; iii) long-horizon reasoning due to the partial observability of complex GCC tasks, which also leads to the demand for long-term memory to maintain past useful experiences; and iv) efficient exploration in a structured manner to discover better strategies and solutions autonomously, *i.e.*, self-improving, which can allow agents to generalize across the various tasks in the digital world.

As shown in Figure 1, we introduce **CRADLE**, a novel modular LMM-powered framework that empowers foundation agents towards GCC. **CRADLE** consists of six key modules: 1) information gathering, to extract the relevant information from multimodal observations; 2) self-reflection, to rethink past experiences about whether the actions and tasks are successfully completed and reasons for possible failures; 3) task inference, to determine whether to continue current tasks or propose a new task given the current situation; 4) skill curation, to generate, update, and retrieve useful skills for the current task; 5) action planning, to generate specific executable operations for keyboard and mouse control via skills; and 6) memory, for storage, summary, and retrieval of past experiences.

As illustrated in Figure 2, tasks in GCC can be broadly divided into two categories: video game playing and software application manipulation. Video games offer the most challenging tasks in GCC due to several key factors. First, the complexity of game environments requires sophisticated problem-solving and adaptive strategies. Second, long-term reasoning is essential to navigate and succeed in these intricate virtual worlds. Third, understanding and mastering new, complex mechanics within games demand rapid learning and cognitive flexibility. Finally, video games test a player’s ability to react quickly and perform precise control and operations, which together create a unique and demanding computational challenge. In addition to the typical embodied control, classical UI manipulation, like menu use and inventory management, is also common during gameplay, which is similar to the other software applications (Raad et al., 2024). Therefore, video games provide rich comprehensive and challenging testbeds to evaluate and improve agents’ various abilities.

In this work, we conduct extensive experiments to demonstrate the generalizability of **CRADLE** in such complex environments, while also mastering diverse everyday software applications in distinct domains. We managed to prove that commercial software is out-of-box testbeds under our framework. The four selected representative games are: epic AAA 3D role-playing game, **RDR2**, 2D pixel-art farming simulation game, **Stardew Valley**, pawn shop simulation game, Dealer’s Life 2, and 3D, top-down view, city-building game, **Cities: Skylines**. The target set of diverse software applications for evaluation includes: **Chrome**, **Outlook**, **CapCut**, **Meitu**, and **Feishu**, as well as one comprehensive software benchmark, **OSWorld** (Xie et al., 2024). We provide a brief introduction to these games in Appendix A, and representative designed tasks for measuring the various abilities of the agent comprehensively in both games and software applications in Appendix Figure 9.

Experimental results show that **CRADLE** exhibits remarkable generalization ability and impressive performance across the four previously unexplored commercial video games, the five target software applications, and the comprehensive contemporaneous **OSWorld** benchmark. To our best knowledge, **CRADLE** is the first to enable LMM-based agents to follow the main storyline and complete one-hour-long real missions in a complex AAA game, **RDR2**. **CRADLE** also manages to create a city with nearly a thousand people in **Cities: Skylines**, farm and harvest parsnips in **Stardew Valley**,

¹This setting can be seamlessly extended to other digital devices, *i.e.*, mobile phones, game controllers, and virtual reality headsets with standard input and output.

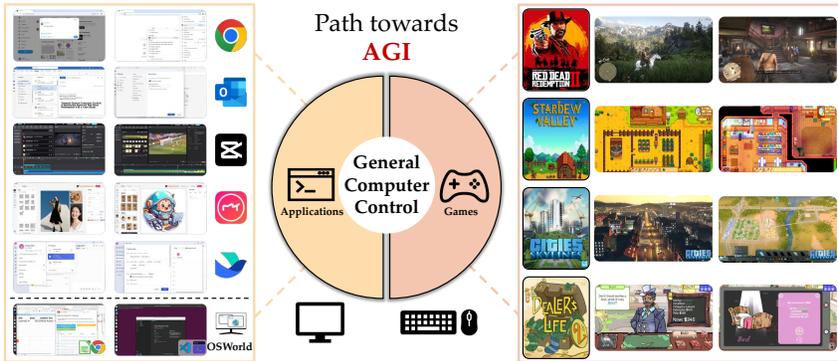


Figure 2: Taxonomy of GCC and the games and software investigated in this work.

trade and bargain with a maximal weekly total profit of 87% in Dealer’s Life 2. Besides, **CRADLE** can not only operate daily software, like Chrome and Outlook, but also edit images and videos using Meitu and CapCut, and perform office tasks in Feishu. Able to interact with software in a unified manner, **CRADLE** greatly extends the reach of AI agents by making it easy to convert any software, especially complex games, into benchmarks to evaluate agents’ various abilities and facilitate further data collection, paving the way for generalism. We hope **CRADLE** can accelerate the development of more powerful foundation agents, thereby advancing the path towards AGI.

2 RELATED WORK

Agents for Software Applications. While previous LLM-based web agents (Deng et al., 2023; Zhou et al., 2023; Gur et al., 2023; Zheng et al., 2024b) show some promising results in effectively interacting with content on webpages, they usually use raw HTML code and DOM tree as input and interact with the available element IDs, ignoring the rich visual patterns with key information, like icons, images, and spatial relations. Multimodal web agents (Hong et al., 2023; Furuta et al., 2023; Yan et al., 2023; He et al., 2024; Zheng et al., 2024a) and mobile app agents (Yang et al., 2023b; Wang et al., 2024b) have also been explored. Though using screenshots as input, they still need to use built-in APIs to get the available interactive element IDs to execute corresponding actions. Several recent works (Cheng et al., 2024; Zhang et al., 2024; Wu et al., 2024; Kapoor et al., 2024) aim to apply web agents to more applications by using keyboard and mouse for control. However, they primarily focus on the static websites and lack the generalizability to other domains.

Agents for Video Games. Several attempts try to develop foundation agents for complex video games, such as Minecraft (Wang et al., 2023b;a; 2024a), Starcraft II (Ma et al., 2023) and Civilization-like game (Qi et al., 2024) with textual observations obtained from internal APIs and pre-defined semantic actions. Although JARVIS-1 (Wang et al., 2023a) claims to interact with the environment in a human-like manner with the screenshots as input and mouse and keyboard for control, its action space is predefined as a hybrid space composed of keyboard, mouse, and API. The game-specific observation and action spaces prohibit the generalization of them to other novel games. SIMA(Raad et al., 2024) trained embodied agents to complete 10-second-long basic tasks over ten 3D video games, and the results are promising to be scaled up.

Due to the space limitation, we provide a detailed discussion of the related work in Appendix B.

3 THE CRADLE FRAMEWORK

To pursue GCC, we propose **CRADLE**, illustrated in Figure 3, a modular and flexible LMM-powered framework that can properly handle the challenges GCC presents. The framework should have the ability to understand and interpret computer screens and dynamic changes between consecutive frames from arbitrary software and be able to generate reasonable computer control actions for precise execution. This suggests that a multimodal model with powerful vision and reasoning capabilities, in addition to rich knowledge of computer UI and control, is a requirement. In this work, we leverage GPT-4o (OpenAI, 2024b) as the framework’s backbone model.

3.1 ENVIRONMENT IO

Observation and Action Space. **CRADLE** only takes a video clip, recording the execution of the last action, as input and outputs keyboard and mouse operations to interact with environments. The observation space is made up of complete screen videos with different lengths. For the action space, it includes all possible keyboard and mouse operations, including `key_press`, `key_hold`, `key_release`, `mouse_move`, and `wheel_scroll`, where keys include both keyboard keys

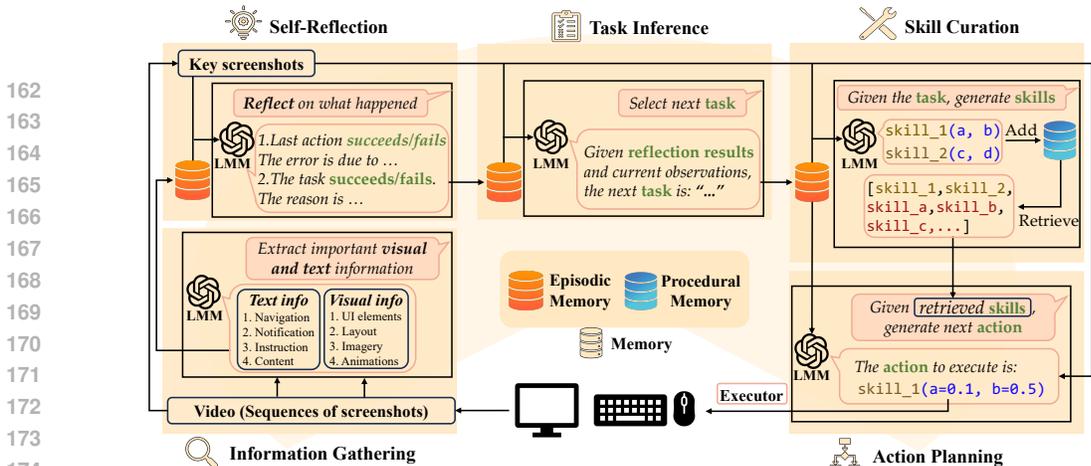


Figure 3: An overview of the **CRADLE** framework. **CRADLE** takes video from the computer screen as input and outputs computer keyboard and mouse control determined through inner reasoning.

and mouse buttons. These operations can be combined in various ways to form combos and shortcuts, execute rapid key sequences, or coordinate timings. We choose to use Python code to simulate these operations and encapsulate them into an `io_env` class.

Information Gathering. Provided with a video clip as input, it is critical for **CRADLE** to capture and extract all useful visual and textual information to understand the recent situation and perform further reasoning. Visual information includes layout, imagery, animations, and UI elements which pose high spatial perception and visual understanding requirements for LMM models. Moreover, we depend on their OCR capabilities to extract textual information in images, which usually includes content (headings and paragraphs), navigation labels (menus and links), notifications, and instructions to convey messages and guide users. For each environment, we enhance LMMs’ abilities with different tools such as template matching (Brunelli, 2009), Grounding DINO (Liu et al., 2023), and SAM (Kirillov et al., 2023) to provide additional grounding for object detection and localization.

Skill and Action Generation As shown in Figure 4, to bridge the gap between semantic actions generated by LMMs and OS-level executable actions, **CRADLE** uses LMMs to generate code functions as semantic-level skills, which encapsulate lower-level keyboard and mouse control. Similar to how humans improve while playing, these skills can be developed from scratch according to in-game tutorials and guidance, game manuals and settings, or through self-exploration as the game progresses. These skills can also be pre-defined or composited to solve more complex tasks. An action usually consists of a single or multiple skills instantiated with any necessary parametric aspects, such as duration, position, and speed. An *Executor* will be triggered to map these semantic actions to the OS-level keyboard and mouse commands to interact with the environment.

3.2 MEMORY

CRADLE stores and maintains all the useful information from the environment or outputted by each module through a memory mechanism, consisting of episodic memory and procedural memory.

Episodic Memory. Episodic memory is used to maintain current and past experiences, including key screenshots from each video observation, and everything useful outputted by LMMs and advanced tools, e.g., textual and visual information, actions, tasks, and reasoning from each module. To facilitate retrieval and storage, periodical summarization is conducted to abstract recently added

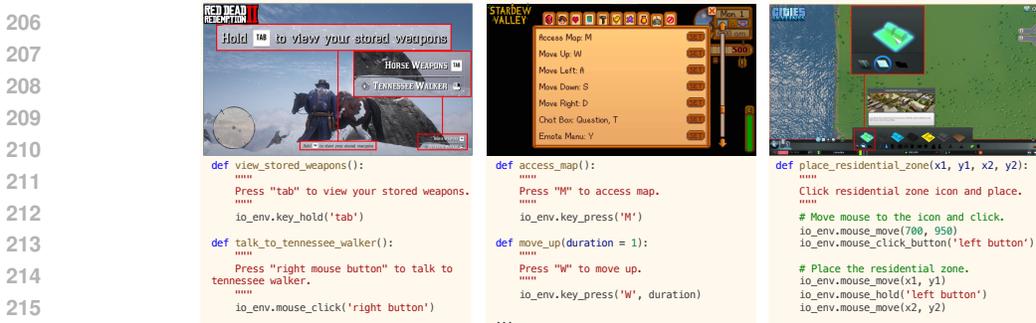


Figure 4: Examples for skill generation according to in-game guidance in RDR2 (left), in-game manual in Stardew Valley (middle), self-exploration in Cities: Skylines (right). Code and comments are shown in brevity.

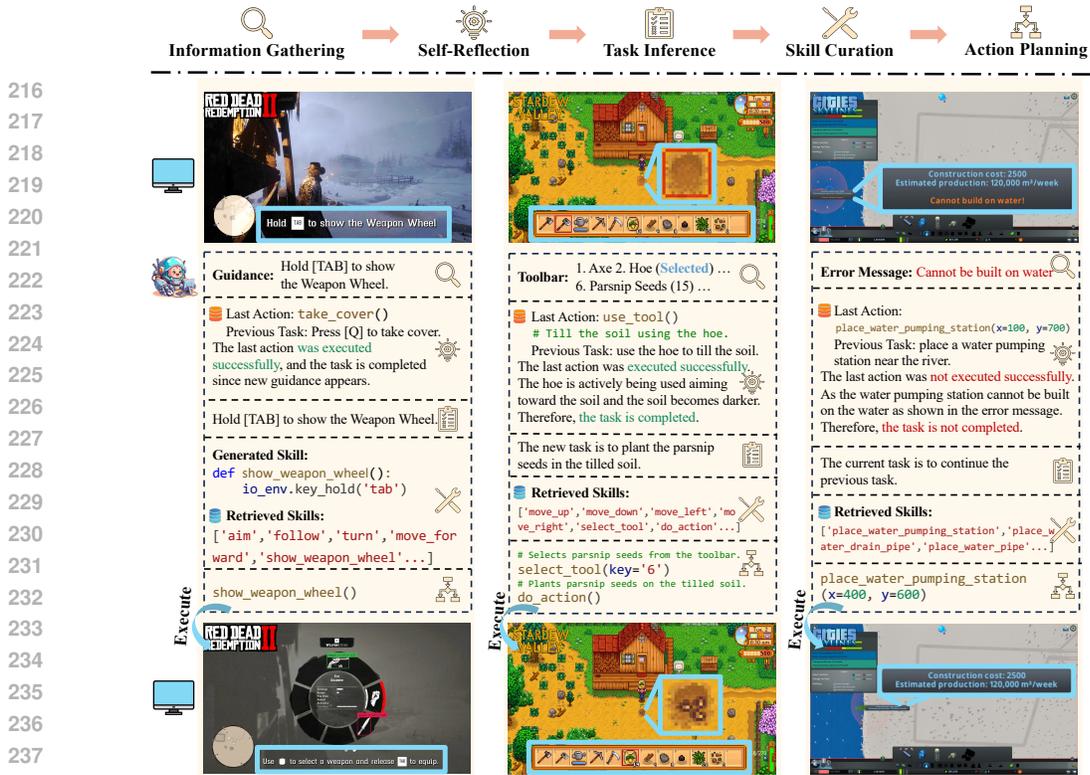


Figure 5: Illustrative examples of **CRADLE**'s complete workflow in RDR2 (left), Stardew Valley (middle) and Cities: Skylines (right). Prompts are shown partially for brevity.

multimodal information into long-term summaries. The incorporation of episodic memory enables **CRADLE** to effectively retain crucial information over extended periods.

Procedural Memory. This memory is specific to storing and retrieving skills in code form, which can be learned from scratch as shown in Figure 4, or pre-defined in procedural memory. Skills can be added, updated, or composed to the procedural memory in the skill curation module. Same as Voyager (Wang et al., 2024a), skills are retrieved according to the similarities between their corresponding embedding and task description.

3.3 REASONING

Based on the extracted information from observations and memory, **CRADLE** conducts high-level reasoning and then makes the next decision. This process is analogous to “**reflect on the past, summarize the present, and plan for the future**”, which is broken down into the following modules.

Self-Reflection. The reflection module initially evaluates whether the last executed action was successfully carried out and whether the task was completed. Sequential key screenshots from the last video observation, along with the previous context for action planning and task inference are fed to the LMM for reasoning. Additionally, we also request the LMM to provide an analysis of any failure. This valuable information enables **CRADLE** to remedy inappropriate decisions or less-than-ideal actions. Furthermore, reflection can also be leveraged to inform re-planning of the task and bring the agent closer to target task completion, better understand the factors that led to previous successes, or suggest how to update or improve specific skills.

Task Inference. After reflecting on the outcome of the last executed action, **CRADLE** needs to analyze the current situation to infer the most suitable task for the current moment. We let LMMs determine the highest priority task to perform and when to stop an ongoing task and start a new one.

Skill Curation. As the task is specified, **CRADLE** needs to prepare the tactics to accomplish it, by retrieving useful skills from the procedural memory, updating existing skills, or generating new ones. The new skill will be stored in the procedural memory for future utilization.

Action Planning. **CRADLE** needs to select the appropriate skills from the curated skill set and instantiate these skills into a sequence of executable actions by specifying any necessary parametric aspects (e.g., duration, position, and target) according to the current task and history information. The generated action is then fed to the *Executor* for interaction with the environment.

4 EMPIRICAL STUDIES

In this section, we first introduce the practical implementation of the current Cradle framework and then present the empirical results of deploying CRADLE across various challenging environments representative of GCC settings, demonstrating its comprehensive capabilities.

4.1 GENERAL IMPLEMENTATIONS

Input. CRADLE applies *gpt-4o-2024-05-13* as backbone. It only takes a video clip, which records the execution progress of the last action, as input. To lower the frequency of interaction with backbone models and reduce the strain on the computer, video is recorded at 2 fps, which proves to be sufficient in most cases for information gathering without missing any important information.

Skills. CRADLE uses Python code to simulate keyboard and mouse operations, which is encapsulated by an `io_env` class to achieve OS-agnostic interface. Skills are generated based on these basic operations. We use OpenAI’s *text-embedding-ada-002 model* (OpenAI, 2022) to generate embeddings for each skill, stored in the procedural memory and retrieved according to the similarities.

Prompts. Prompts used by each module are initialized by the corresponding templates in Markdown-style format. These prompt templates provide a minimal workflow with basic rules for the module to run and use placeholders of each key for input and output. CRADLE automatically retrieves the corresponding value for each key in the input from the episodic memory and forms valid requests to query LMMs with the values and templates. After receiving responses from LMMs, CRADLE automatically extracts the keys in the output and stores them in the episodic memory. Users can freely customize their own prompts without writing any code.

Apply to new environments. Theoretically, CRADLE can be directly deployed to new video games or other software applications with the default prompt templates and empty procedural memory. Due to the limited ability of current LMMs and the complexity of challenging environments and tasks, prompt engineering may need to be applied to every module to enhance LMMs’ reasoning ability and introduce domain knowledge. Additional tools can also be applied to provide extra grounding and domain knowledge as part of the prompt input. Procedural memory can be initialized with hand-craft skills to mitigate the incomplete tutorials provided by the software and the complexity of tasks. Users may need to analyze the task-specific issue and choose a suitable solution. We provide all the implementation details and prompts we use for each software in Appendices D to K.

Experimental Settings. If not specifically mentioned, all experiments are conducted in five runs under a maximum step limit. For each video game, we hired five human players, who never played the corresponding game before, to do the evaluation. Before they start the experiments, they will read the prompts used by Cradle agents for fair comparison. Every player played the task once. We apply human evaluation to all tasks, except for OSWorld, which provides automatic evaluation scripts. Estimated experimental cost of the time and API usage is provided in Appendix C.

Task Introduction. As shown in Figure 6 and 7, for **RDR2**, we mainly focus on evaluating agents on the first two complete missions of the main storyline in Chapter I, which can be divided into 13 tasks according to the in-game checkpoints, including but not limited to navigation, NPC interaction, inventory management, house exploration, and combat. It usually takes a human player about an hour to complete these missions. Few previous studies tackle such long-duration tasks and rich semantic environments. It is an ideal scenario to emulate a novice player learning to play the game from scratch according to the rich in-game tutorials and hints. For **Stardew Valley**, we propose three essential tasks at the stage of the game, *i.e.*, *Farm Cleanup*: Clear the obstacles on the farm, such as weeds, stones, and trees, as much as possible to prepare for farming; 2) *Cultivation*: Plant the parsnip seed, water every day and harvest at least one mutual parsnip; 3) *Shopping*: Go to the general store in the town, which is out of the scope of the current map, to buy more seeds and return home. For **Dealer’s Life**, the agent is tasked with managing a pawn shop for a week, appraising item values and haggling with the customers to secure deals. For **Cities: Skylines**, the task is to build a reasonable city ending in as much population as possible, with the initial starting funds of €70,000, and basic road, water and electricity facilities. Moreover, we define five representative domain-specific tasks for each of the five **Software Applications** in our diverse target set. We provide an overview of all the tasks for both games and software applications in Appendix Figure 9.

4.2 PERFORMANCE ACROSS ENVIRONMENTS

Red Dead Red Redemption 2. Figure 6 shows that CRADLE can efficiently complete simple navigation tasks with a few steps like following an NPC or going to specific locations on the ground (*e.g.*, *Follow Dutch*, *Go to Town* and *Go to Barn*). Another following task, *Follow Javier*, and the searching task, *Search John*, are dangerous for the rugged and winding path up to the snow

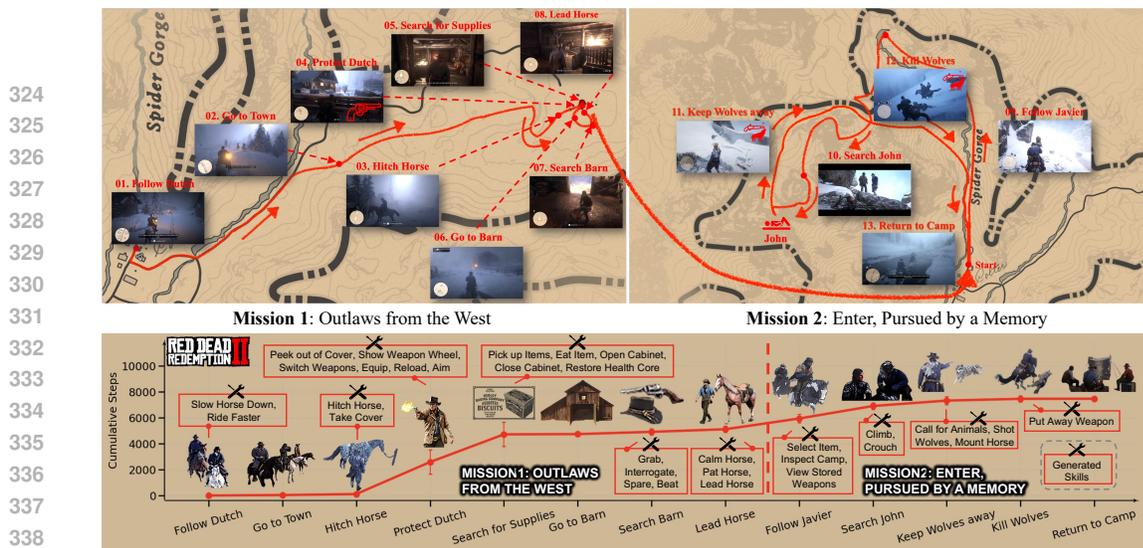


Figure 6: The first row demonstrates the trajectory of 13 sequential tasks in the two main storyline missions. The second row shows the cumulative steps **CRADLE** takes to complete each task in the two missions, starting from the beginning of the game. If a task fails, **CRADLE** can select the ‘retry checkpoint’ option to retry the task. Skills generated during the task completion are also illustrated in the figure. We only provide key skills for brevity. Error bars represent the standard deviation of steps needed to complete each task separately.

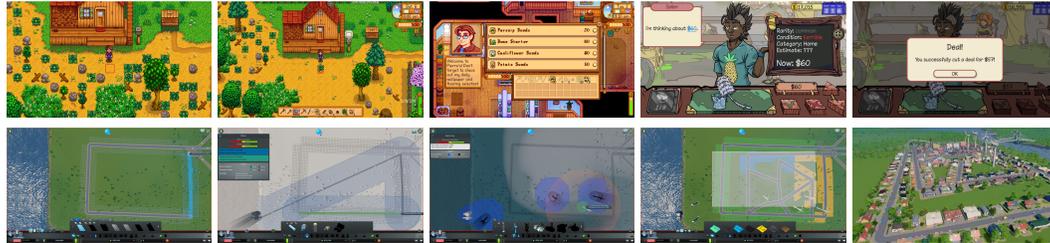


Figure 7: The first row sequentially shows farm clearup, cultivation and shopping in Stardew Valley and haggling and deal in Dealer’s Life 2. The second row sequentially shows road construction, water pipe laying, wind turbine building, zoning and the display of the city built by **CRADLE** in Cities: Skylines.

mountain with cliffs. Note that Cradle is able to retry the checkpoint automatically according to the game guidance if the task fails. Therefore, **CRADLE** takes more steps for retrying the task in these dangerous areas. In addition, Cradle spends about one-fourth of the total steps in the task of *Protect Dutch*, which is a long-horizontal task with nighttime combat. Many key skills are generated in this task for weapon management and shooting movement. The visibility is poor due to the snow falling in the dark, preventing GPT-4o from accurately recognizing and locating enemies or objects and precisely timing decisions, even equipped with Grounding DINO as an additional detection tool. More times of retry, combined with the need for frequent interactions during combat and the long horizon of the task, lead to this task requiring a large number of steps to complete. The success rate of the combat has significantly improved during the day with much fewer steps for completion, as shown by tasks like *Keep Wolves away*. Additionally, indoor tasks like *Search for Supplies* are also challenging due to GPT4-o’s limited spatial perception, which finds it difficult to locate target objects and ends up circling aimlessly around the house. Moreover, the room contains numerous interactive items unrelated to the task, resulting in much more steps for the agent to complete the task. Overall, **CRADLE** requires approximately 8,000 steps to complete both missions, taking around 98 minutes of in-game time, compared to the average of 67 minutes for human players. It is the first time for LMM-powered AI agents to exhibit comparable performance in complex AAA games.

Stardew Valley. As shown in Table 1, we surprisingly find that GPT-4o struggles with accurately recognizing and locating objects near the player in this pixel-art game. This leads to difficulties for the agent to interact with objects or people, as it requires the player to stand precisely in front of them in the grid (e.g., when entering doors, using a pickaxe to break stones). It explains the inefficiency in the farming task though the agent manages to clear up most of the obstacles in front of the house within 100 steps and poor performance in the shopping task. On the other hand, relying on episodic summarization and task inference, **CRADLE** manages to obtain the parsnip by watering the seed for four days and harvesting. Given GPT-4’s limited visual capabilities in this game, there is still room for improvement in narrowing the gap between **CRADLE** and human players.

Dealer’s Life 2. Table 1 shows that **CRADLE** demonstrates robust performance and efficient profit-making on the *Weekly Shop Management* task, successfully finalizing 93.6% of potential transactions, with an average of 2 negotiation rounds per customer, and generally aiming for a profit rate of over 50% at the initial offer. It consistently generates profit across all runs, maintaining a total profit rate of +39.6%, peaking at +87.4% in a single run. In this game, **CRADLE** significantly outperforms human players. The achievements are mainly attributed to its cautious strategy, by bargaining within a smaller range of price variation but haggling more frequently, resulting in a significantly higher turnover rate. In contrast, human players usually fail the deal due to their aggressive strategy by proposing an unreasonable price and sometimes confusing buying and selling.

Cities: Skylines. Table 1 shows that **CRADLE** is able to complete most of the city design with the averaged maximal population of 450 and the highest single population exceeding 860. **CRADLE** manages to build the roads in a closed loop to ensure smooth traffic flow, place multiple wind turbines to provide sufficient electricity supply and cover more than 90% of available area with residential, commercial and industrial zones, but fails to provide sufficient water supply for all the regions reliably. The most common failure arises from the missing of water pipes. **CRADLE** often fail to connect them with each other to cover all zones, resulting in localized water shortages in the city, and preventing new residents from moving in. The issue also arises from GPT-4o’s limited visual understanding, making it difficult to accurately recognize which areas are already covered by the water pipes. We empirically observed that these mistakes usually could be fixed within three unit operations (building or removing a road/facility/a place of zones is counted as one unit operation). Then cities built by **CRADLE** can eventually reach a population of more than one thousand. We provide a detailed case study in Appendix H.5.2. Overall, as shown in Table 1, without the manual fixes, **CRADLE** still beats human players even though it suffers from local water storage. Human players typically pay insufficient attention to budget management and tend to allocate a disproportionate amount of funds to the construction of wind turbines for electricity, resulting in limited road construction and residential areas to attract residents.

Software Applications. Figure 8 shows **CRADLE**’s performance across tasks on five applications. Multiple tasks remain challenging. Even with a well-known GUI, like Chrome and Outlook, GPT-4o still cannot recognize specific UI items to interact with and also struggles with visual context. For example, forgetting to press the Save button in an open dialog, or not distinguishing between a nearby enabled button vs. a distant and disabled one (e.g., when posting on Twitter). The phenomenon is more severe in the UI with non-standard layouts, like CapCut, Meitu, and Feishu. Lacking prior knowledge by GPT-4o leads to the failure of task inference and selecting the correct skills.

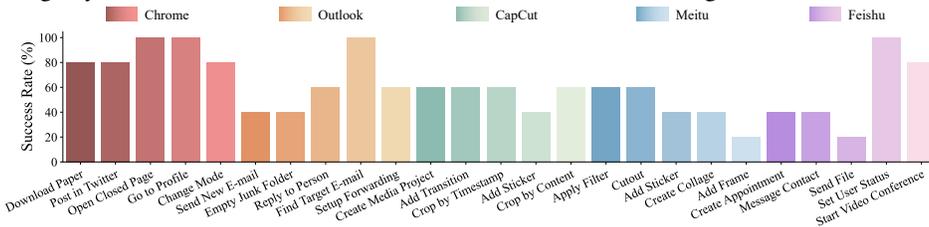


Figure 8: Cradle’s performance in software applications. Each task is run for 5 trials.

OSWorld. Table 2 shows that **CRADLE** achieves the overall highest success rate in OSWorld, compared to the baselines without relying on any internal APIs to provide extra grounding labels, e.g., Set-of-Mark (SoM) (Yang et al., 2023a). The information gathering module improves ground-

Table 1: **CRADLE**’s and human players’ performance in Stardew Valley, Dealer’s Life 2 and Cities: Skylines with each trial run for at most 100, 500, 1000 steps respectively. $\frac{1}{5}$ indicates one successful run out of five runs.

Stardew Valley		
Task	Cradle	Human
Farm Cleanup (Grids Num.)	14.8 ± 5.0	35.2 ± 14.5
Cultivation	4/5	5/5
Shopping	1/5	5/5
Dealer’s Life 2		
Metrics	Cradle	Human
Avg. Haggling Count	1.95 ± 0.43	1.63 ± 0.53
Turnover Rate (%)	93.6 ± 6.9	68.4 ± 22.2
Item Profit Rate (%)	37.8 ± 19.1	21.1 ± 13.6
Total Profit Rate (%)	39.6 ± 27.3	17.3 ± 15.1
Cities: Skylines		
Metrics	Cradle	Human
Closed-loop Road	4/5	5/5
Water Supply	1/5	3/5
Power Supply	5/5	5/5
Zoning Coverage	4/5	4/5
Population	450 ± 224	415 ± 416

Table 2: Success rates (%) of different methods in OSWorld.

Method	Office (117)	OS (24)	Daily (78)	Workfl-ow(101)	Professi-onal (49)	All (369)
GPT-4o	3.58	8.33	6.07	5.58	4.08	5.03
GPT-4o+SoM	3.58	20.83	3.99	3.60	2.04	4.59
CRADLE	3.58	16.67	6.55	5.48	20.41	7.81

ing for more precise action execution, increasing the performance. The self-reflection module enables Cradle to predict infeasible tasks and subsequently fix mistakes, shown in the Professional domain results, where it achieves a 20.41% success rate, significantly surpassing the baselines.

4.3 BASELINE COMPARISON

Since no existing methods are fully applicable to the GCC setting, we select several representative methods with necessary adaptations to make them applicable to GCC, labeling them as "like" in Table 3. Compared to **CRADLE**, React (Yao et al., 2023)-like method only has gather information, skill curation, action planning and procedural memory module, while Reflexion (Shinn et al., 2023)-like method adds a self-reflection and episodic memory, compared to React-like. To show the necessity of multimodal input without access to APIs, we let GPT-4o describe the image and then feed the textual description to Voyager (Wang et al., 2024a)-like as input. Additionally, experiments with GPT-4o and Claude 3 Opus (Anthropic, 2024) as backbone are conducted. Due to the limitation of requests per minute, other prompting methods like self-consistency (Wang et al., 2022) and TOT (Yao et al., 2024) are not considered. Note that methods here refer to the agents initialized by the corresponding framework with game-specific implementations.

Table 3: **Baseline comparison** for five task in RDR2 and one task in Stardew Valley (*Cultivation*). Numbers before the brackets are average steps for completion. N/A indicates failure for all trials. Every task is run 5 times. Each trial is run for at most 500 steps in RDR2 and 100 steps in Stardew Valley.

Method	Follow Dutch	Follow Micah	Hitch Horse	Protect Dutch	Search for Supplies	Cultivation
React-like (GPT-4o)	15 ± 2 (5/5)	74 ± 0 (1/5)	N/A	N/A	N/A	N/A
Reflexion-like (GPT-4o)	19 ± 4 (5/5)	58 ± 14 (2/5)	N/A	N/A	N/A	N/A
Voyager-like (GPT-4o)	32 ± 12 (3/5)	N/A	N/A	N/A	N/A	N/A
CRADLE (Claude 3 Opus)	30 ± 7 (5/5)	52 ± 17 (4/5)	N/A	N/A	N/A	N/A
CRADLE (GPT-4o) (Ours)	13 ± 3 (5/5)	33 ± 3 (5/5)	26 ± 5 (4/5)	461 ± 0 (1/5)	134 ± 0 (1/5)	24 ± 4 (4/5)

As shown in Table 3, all the baseline methods can only complete simple and straightforward tasks without complex targets and time delays. Compared to React-like method, Reflexion-like method has better performance in the task of *Follow Micah* and still fails to complete more complex tasks, emphasizing the importance of task inference and procedural memory. Voyager-like method that loses vision suffers to accomplish tasks and are the worst of all comparison methods. **CRADLE** with GPT-4o always has the best performance across all tasks. **CRADLE** with GPT-4o has the best performance, while Claude 3 Opus fails frequently due to unreliable OCR ability of the guidance, leading to incorrect skill generation and failures of complex tasks.

Figure 4 provides the detailed performance of each baseline method in the *Cultivation* task in Stardew Valley. Without task inference and episodic memory for summarization, even React-like and Reflexion-like methods sometimes managed to get the parsnip to sprout from the ground, they failed to harvest it because GPT-4o failed to recognize the mature parsnip. Episodic memory can help **CRADLE** record the days of watering and know when the crop can be harvested. Voyager-like method struggles with getting out of the house and returning home due to the lack of visual input. Claude 3 Opus also has difficulties in localizing the position of the character and the crop. Moreover, it prefers moving characters much more frequently than GPT-4, resulting in the failure to position the character in front of the crop.

4.4 ABLATION STUDY

Besides comparing with other baseline methods, we provide a complete ablation study by systematically removing each module of Cradle to show the effectiveness in Table 5. We mainly show the results of 6 consecutive subtasks at the beginning of the main storyline, separated from the tasks of *Follow Micah*, *Hitch Horse* and *Protect Dutch* in RDR2. Note that the combination of skill curation, action planning and procedural memory is the minimal unit of our framework. Without any of them, the agent cannot generate and execute valid actions successfully. So these modules are not ablated.

The most significant decline in agent capabilities arises from the absence of the information gathering module. Without this module, the agent is unable to extract key information in the observation,

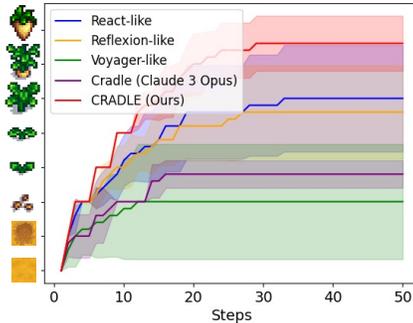


Table 4: Performance of each method in task *Cultivation*. The Y-axis shows the stage of parsnip. Only if the mature parsnip (shown on the top of the y-axis) is obtained will this trial be counted as a success.

which is critical for all other modules to function effectively. The second largest impact comes from the lack of the self-reflection module, which is instrumental in correcting mistakes and recognizing when the agent is stuck, such as in the subtask of *Go to Shed*. Third, the task inference module is vital for tasks that require strict adherence to guidance, like *Switch Weapon*. In these cases, the in-game instructions appear only at the beginning of the task, as seen in *Follow Micah* and *Go to Shed*. Lastly, episodic memory becomes increasingly important as tasks grow more complex, requiring more steps to complete, such as in *Go to Shed* and *Combat*, which involve far more steps than other subtasks. Overall, each module plays a crucial and distinct role in the Cradle framework. Removing or isolating any of them significantly reduces the agent’s effectiveness, underscoring the importance of their integrated function.

Table 5: Success rates of each variant by systematically removing Cradle’s module on six consecutive subtasks in RDR2. Every subtask is run 5 times. Each of subtasks are run for at most 500 steps.

Subtask	w/o Information Gathering	w/o Self-Reflection	w/o Task Inference	w/o Episodic Memory	CRADLE
Follow Micah	0%	0%	40%	80%	100%
Hitch Horse	0%	100%	100%	100%	100%
Go to Shed	0%	20%	40%	20%	80%
Peek out of Cover	60%	100%	80%	100%	100%
Switch Weapon	0%	80%	60%	80%	100%
Combat	0%	0%	0%	0%	20%

5 LIMITATIONS AND FUTURE WORK

Despite **CRADLE**’s encouraging performance across games and software, several limitations remain. i) Due to the limited ability of current LMM models, **CRADLE** struggles in recognizing out-of-distribution (OOD) icons and completing OOD tasks, such as games with non-realistic styles, *i.e.*, *Stardew Valley*. As LMMs evolve, they can further improve **CRADLE**’s performance. ii) Another general bottleneck for LMM-based agents is the latency caused by the limited inference speed of LMMs, which can also be alleviated as LMMs evolve (*e.g.*, Realtime API (OpenAI, 2024a)). iii) Audio, as an important modality, often plays an important role in games and software; which has not been considered in this work. The future work will be enabling **CRADLE** to process the audio and graphical input simultaneously. iv) [As the preliminary attempt towards GCC](#), most **CRADLE**’s modules need to call LMM explicitly to process the input for best performance, resulting in frequent interactions with LMM and potentially high costs and long delays. The six modules represent a problem-solving mindset; as LMM capabilities improve, some or even all of these modules may be combined into a single request. [Exploring other potential GCC frameworks is also promising.](#) v) In this work, we mainly focus on enabling foundation agents to interact with various software in a unified manner without taking training into consideration. As SIMA (Raad et al., 2024) has already shown promising results in a similar setting with trained agents, we will let **CRADLE** autonomously explore and improve over environments through RL (Tan et al., 2023) or collect expert demonstrations for supervised learning (Raad et al., 2024). vi) Though **CRADLE** is broadly applicable to any computer task, only a few selected tasks are investigated in this work. We plan to expand its application to a wider range of targets, delve deeper into complex games, and enhance its adaptability for users. [vii\) Due to the large scope of the experiments conducted in this work, the number of runs for each task and human participants are limited. A more comprehensive evaluation can be beneficial.](#) **CRADLE** holds great potential to improve effective general computer task completion and boost research and deployment of foundation agents. However, there is also a risk of unintended or unsuitable usage, including developing game cheats, incorrect operations of software with harmful failures, or other negative agent behavior. Therefore, additional regulations or safeguards are required for secure and responsible deployments across digital and physical environments.

6 CONCLUSION

We introduce GCC, a general and challenging setting to control diverse video games and software with a unified and standard interface, paving the way towards general foundation agents across all digital world tasks. To properly address the challenges GCC presents, we propose a novel framework, **CRADLE**, which exhibits strong performance in reasoning and performing actions to accomplish various missions in a set of complex video games and common software applications. To the best of our knowledge, **CRADLE** is the first framework that enables foundation agents to succeed in such a diverse set of environments without relying on any built-in APIs. The success of **CRADLE** greatly extends the reach of foundation agents and demonstrates the feasibility of converting any software, especially complex games, into benchmarks to evaluate agents’ general intelligence and facilitate further data collection for self-improvement. Although **CRADLE** still faces difficulties in certain tasks, it serves as a pioneering work to develop more powerful LMM-based agents towards GCC, combining both further framework enhancements and new advances in LMMs.

REFERENCES

- 540
541
542 Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.
543
544
- 545 Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon
546 Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching
547 unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654,
548 2022.
- 549 Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew
550 Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by com-
551 bining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.
- 552 Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade learning envi-
553 ronment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*,
554 47:253–279, 2013.
- 555 Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębniak, Christy
556 Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large
557 scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- 558 Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choroman-
559 ski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. RT-2: Vision-language-action
560 models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023a.
- 561 Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho,
562 Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as I can, not as I say: Grounding
563 language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. PMLR, 2023b.
- 564 Roberto Brunelli. *Template matching techniques in computer vision: theory and practice*. John
565 Wiley & Sons, 2009.
- 566 Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca
567 Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural
568 information processing systems*, 32, 2019.
- 569 Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong
570 Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. *arXiv preprint
571 arXiv:2401.10935*, 2024.
- 572 Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia,
573 Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning
574 with a human in the loop. In *International Conference on Learning Representations*, 2019. URL
575 <https://openreview.net/forum?id=rJeXCo0cYX>.
- 576 Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson
577 Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Proctor: Large-scale embodied
578 ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–
579 5994, 2022.
- 580 Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and
581 Yu Su. Mind2Web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*,
582 2023.
- 583 Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter,
584 Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multi-
585 modal language model. *arXiv preprint arXiv:2303.03378*, 2023.
- 586 Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan,
587 Jakob Nicolaus Foerster, and Shimon Whiteson. SMACv2: An improved benchmark for coop-
588 erative multi-agent reinforcement learning. In *Thirty-seventh Conference on Neural Information
589 Processing Systems Datasets and Benchmarks Track*, 2023. URL <https://openreview.net/forum?id=50jLGiJW3u>.

- 594 Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang,
595 De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied
596 agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:
597 18343–18362, 2022.
- 598 Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin
599 Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint*
600 *arXiv:2305.11854*, 2023.
- 601 Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen
602 Zhang, Peiyi Wang, Xiangwu Guo, et al. ASSISTGUI: Task-oriented desktop graphical user
603 interface automation. *arXiv preprint arXiv:2312.13108*, 2023.
- 604 Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. Vrkitchen: an
605 interactive 3d virtual environment for task-oriented learning. *arXiv preprint arXiv:1903.05757*,
606 2019.
- 607 Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and
608 Aleksandra Faust. A real-world webagent with planning, long context understanding, and pro-
609 gram synthesis. *arXiv preprint arXiv:2307.12856*, 2023.
- 610 William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela
611 Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of Minecraft demonstrations.
612 *arXiv preprint arXiv:1907.13440*, 2019.
- 613 Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan,
614 and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models.
615 *arXiv preprint arXiv:2401.13919*, 2024.
- 616 Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan
617 Wang, Yuxiao Dong, Ming Ding, et al. CogAgent: A visual language model for GUI agents.
618 *arXiv preprint arXiv:2312.08914*, 2023.
- 619 Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan
620 Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through
621 planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.
- 622 Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia
623 Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-
624 level performance in 3D multiplayer games with population-based reinforcement learning. *Sci-*
625 *ence*, 364(6443):859–865, 2019.
- 626 Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo platform for artifi-
627 cial intelligence experimentation. In *Ijcai*, pp. 4246–4247, 2016.
- 628 Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh,
629 and Ruslan Salakhutdinov. OmniACT: A dataset and benchmark for enabling multimodal gener-
630 alist autonomous agents for desktop and web, 2024.
- 631 Christian Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018. URL <https://github.com/Kautenja/gym-super-mario-bros>.
- 632 Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete
633 Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceed-*
634 *ings of the IEEE/CVF International Conference on Computer Vision*, pp. 4015–4026, 2023.
- 635 Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham
636 Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating
637 multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.
- 638 Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Car-
639 los Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research
640 football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on*
641 *artificial intelligence*, pp. 4501–4510, 2020.

- 648 Joel Z Leibo, Edgar A Dueñez-Guzman, Alexander Vezhnevets, John P Agapiou, Peter Sunehag,
649 Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. Scalable eval-
650 uation of multi-agent reinforcement learning with melting pot. In *International conference on*
651 *machine learning*, pp. 6187–6199. PMLR, 2021.
- 652 Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen,
653 Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simula-
654 tion for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021.
- 656 Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement
657 learning on web interfaces using workflow-guided exploration. In *International Conference on*
658 *Learning Representations (ICLR)*, 2018. URL <https://arxiv.org/abs/1802.08802>.
- 659 Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei
660 Yang, Hang Su, Jun Zhu, et al. Grounding Dino: Marrying dino with grounded pre-training for
661 open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.
- 663 Weiyu Ma, Qirui Mi, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. Large
664 language models play StarCraft II: Benchmarks and a chain of summarization approach. *arXiv*
665 *preprint arXiv:2312.11865*, 2023.
- 666 Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana
667 Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habi-
668 tat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Con-*
669 *ference on Computer Vision (ICCV)*, 2019.
- 671 Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas
672 Scialom. GAIA: a benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.
- 673 Meredith Ringel Morris, Jascha Sohl-dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Alek-
674 sandra Faust, Clement Farabet, and Shane Legg. Levels of AGI: Operationalizing progress on the
675 path to AGI. *arXiv preprint arXiv:2311.02462*, 2023.
- 677 Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and
678 Qi Wang. ScreenAgent: A vision language model-driven computer control agent. *arXiv preprint*
679 *arXiv:2402.07945*, 2024.
- 680 OpenAI. Universe, 2016. URL <https://openai.com/index/universe/>.
- 682 OpenAI. New and improved embedding model, 2022. URL [https://openai.com/index/
683 new-and-improved-embedding-model/](https://openai.com/index/new-and-improved-embedding-model/).
- 685 OpenAI. Introducing the realtime api, 2024a. URL [https://openai.com/index/
686 introducing-the-realtime-api/](https://openai.com/index/introducing-the-realtime-api/).
- 687 OpenAI. Hello gpt-4o, 2024b. URL <https://openai.com/index/hello-gpt-4o/>.
- 689 Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and
690 Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings*
691 *of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.
- 692 Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Tor-
693 ralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE*
694 *conference on computer vision and pattern recognition*, pp. 8494–8502, 2018.
- 696 Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Part-
697 sey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A
698 co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.
- 699 Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan
700 Zhong, Xiaoyuan Zhang, Zhaowei Zhang, et al. CivRealm: A learning and reasoning odyssey in
701 Civilization for decision-making agents. In *ICLR*, 2024.

- 702 Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton,
703 Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable
704 agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.
705
- 706 Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the
707 wild: A large-scale dataset for Android device control. *arXiv preprint arXiv:2307.10088*, 2023.
708
- 709 Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas
710 Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson.
711 The Starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.
- 712 Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia
713 Pérez-D’Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchammi, et al. igibson 1.0: a simu-
714 lation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International
715 Conference on Intelligent Robots and Systems (IROS)*, pp. 7520–7527. IEEE, 2021.
716
- 717 Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An
718 open-domain platform for web-based agents. In *International Conference on Machine Learning*,
719 pp. 3135–3144. PMLR, 2017.
- 720 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Re-
721 flexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on
722 Neural Information Processing Systems*, 2023. URL [https://openreview.net/forum?
723 id=vAE1hFcKW6](https://openreview.net/forum?id=vAE1hFcKW6).
724
- 725 Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre,
726 Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Von-
727 drus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen
728 Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to
729 rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
730
- 731 Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowl-
732 edge comes from practice: Aligning large language models with embodied environments via
733 reinforcement learning. In *ICLR*, 2023.
- 734 Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M
735 Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. AlphaStar: Mas-
736 tering the real-time strategy game Starcraft II. *DeepMind blog*, 2:20, 2019.
737
- 738 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan,
739 and Anima Anandkumar. Voyager: An open-ended embodied agent with large language mod-
740 els. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL [https:
741 //openreview.net/forum?id=ehfRiF0R3a](https://openreview.net/forum?id=ehfRiF0R3a).
- 742 Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Ji-
743 tao Sang. Mobile-Agent: Autonomous multi-modal mobile device agent with visual perception.
744 *arXiv preprint arXiv:2401.16158*, 2024b.
745
- 746 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdh-
747 ery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models.
748 *arXiv preprint arXiv:2203.11171*, 2022.
- 749 Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin,
750 Zhaofeng He, Zilong Zheng, Yaodong Yang, and Yitao Liang. Jarvis-1: Open-world multi-task
751 agents with memory-augmented multimodal language models. *arXiv preprint arXiv:2311.05997*,
752 2023a.
753
- 754 Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and
755 select: Interactive planning with large language models enables open-world multi-task agents. In
ICML, 2023b.

- 756 Sarah A. Wu, Rose E. Wang, James A. Evans, Joshua B. Tenenbaum, David C. Parkes, and Max
757 Kleiman-Weiner. Too many cooks: Coordinating multi-agent collaboration through inverse plan-
758 ning. *Topics in Cognitive Science*, n/a(n/a), 2021. doi: <https://doi.org/10.1111/tops.12525>. URL
759 <https://onlinelibrary.wiley.com/doi/abs/10.1111/tops.12525>.
760
- 761 Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao
762 Yu, and Lingpeng Kong. OS-copilot: Towards generalist computer agents with self-improvement.
763 *arXiv preprint arXiv:2402.07456*, 2024.
- 764 Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian,
765 Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al.
766 Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):
767 223–228, 2022.
- 768 Yuchen Xiao, Weihao Tan, and Christopher Amato. Asynchronous actor-critic for multi-agent rein-
769 forcement learning. *Advances in Neural Information Processing Systems*, 35:4385–4400, 2022.
770
- 771 Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing
772 Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal
773 agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*,
774 2024.
- 775 An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang,
776 Yiwu Zhong, Julian McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. GPT-4V in won-
777 derland: Large multimodal models for zero-shot smartphone GUI navigation. *arXiv preprint*
778 *arXiv:2311.07562*, 2023.
- 779 Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark
780 prompting unleashes extraordinary visual grounding in gpt-4v, 2023a.
781
- 782 Zhao Yang, Jiakuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. AppAgent:
783 Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023b.
784
- 785 Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable
786 real-world web interaction with grounded language agents. *Advances in Neural Information Pro-*
787 *cessing Systems*, 35:20744–20757, 2022.
- 788 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
789 ReAct: Synergizing reasoning and acting in language models. In *International Conference on*
790 *Learning Representations (ICLR)*, 2023.
- 791 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik
792 Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Ad-*
793 *vances in Neural Information Processing Systems*, 36, 2024.
794
- 795 Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei
796 Lin, Saravan Rajmohan, et al. UFO: A UI-focused agent for Windows OS interaction. *arXiv*
797 *preprint arXiv:2402.07939*, 2024.
- 798 Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web
799 agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024a.
800
- 801 Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar
802 prompting with memory for computer control. In *ICLR*, 2024b.
- 803 Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
804 Yonatan Bisk, Daniel Fried, Uri Alon, et al. WebArena: A realistic web environment for building
805 autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.
806
807
808
809

Appendix

Table of Contents

810		
811		
812		
813		
814		
815		
816		
817		
818	A Game & Task Introduction	18
819		
820	B Extended Related Work	18
821	B.1 Environments and Benchmarks for Computer Control	18
822	B.2 LMM-based Agents for Computer Tasks	19
823		
824	C Experimental Cost	20
825		
826	D General Implementation	20
827		
828	E Red Dead Redemption II	23
829	E.1 Introduction to RDR2	23
830	E.2 Objectives	23
831	E.3 Implementation Details	24
832	E.4 Case Studies	29
833	E.5 Limitations of GPT-4o and GPT-4V	32
834		
835	F Stardew Valley	35
836	F.1 Introduction to Stardew Valley	35
837	F.2 Objectives	35
838	F.3 Implementation Details	35
839	F.4 Case Studies	38
840	F.5 Limitations of GPT-4o	39
841		
842	G Dealer’s Life 2	40
843	G.1 Introduction to Dealer’s Life 2	40
844	G.2 Objectives	40
845	G.3 Implementation Details	40
846	G.4 Case Studies	42
847	G.5 Quantitative Evaluation	43
848	G.6 Evaluation Metrics	43
849		
850	H Cities: Skylines	45
851	H.1 Introduction to Cities: Skylines	45
852	H.2 Objectives	46
853	H.3 Evaluation Metric	46
854	H.4 Implementation Details	47
855	H.5 Case Studies	50
856		
857	I Software Applications	52
858	I.1 Selected Software Applications	52
859	I.2 Software Tasks	52
860	I.3 Quantitative Evaluation	56
861	I.4 Implementation Details	56
862	I.5 Case Studies	61
863	I.6 Limitations of GPT-4o	63
	J OSWorld	64
	J.1 Introduction to OSWorld	64

864	J.2 OSWorld Tasks	64
865	J.3 Implementation Details	65
866	J.4 Application Target and Setting Challenges	67
867	J.5 Case Studies	67
868	J.6 Quantitative Evaluation	69
869		
870	K Cradle Prompts	69
871	K.1 Prompts for RDR2	69
872	K.2 Prompts for Cities: Skylines	78
873	K.3 Prompts for Stardew Valley	85
874	K.4 Prompts for Dealer’s Life 2	104
875	K.5 Prompts for Software Applications	109
876		
877		
878		
879		
880		
881		
882		
883		
884		
885		
886		
887		
888		
889		
890		
891		
892		
893		
894		
895		
896		
897		
898		
899		
900		
901		
902		
903		
904		
905		
906		
907		
908		
909		
910		
911		
912		
913		
914		
915		
916		
917		

Environments and Benchmarks on Video Games. On the other side, many research environments are built on top of video games, significantly advancing the study of decision-making, especially, reinforcement learning (RL). Examples include but are not limited to Atari games (Bellemare et al., 2013), Super Mario Bros (Kauten, 2018), Google Research Football (Kurach et al., 2020), Minecraft (Johnson et al., 2016; Guss et al., 2019; Fan et al., 2022), Dota II (Berner et al., 2019), StarCraft II (Vinyals et al., 2019; Samvelyan et al., 2019; Ellis et al., 2023), Quake III (Jaderberg et al., 2019), Gran Turismo (Wurman et al., 2022), Diplomacy (Bakhtin et al., 2022) and Civilization (Qi et al., 2024). Additionally, many custom-built environments, especially grid world and embodied scenarios, are created from scratch in a game-like manner to facilitate agent development, such as BabyAI (Chevalier-Boisvert et al., 2019), Melting Pot (Leibo et al., 2021), Overcooked (Carroll et al., 2019; Wu et al., 2021; Xiao et al., 2022), VRKitchen (Gao et al., 2019), VirtualHome (Puig et al., 2018), iGibson (Shen et al., 2021; Li et al., 2021), ProcTHOR (Deitke et al., 2022), Habitat (Manolis Savva* et al., 2019; Szot et al., 2021; Puig et al., 2023), and Generative agents (Park et al., 2023).

Each of these environments highly relies on the accessibility of the open-source code or provided built-in APIs. Significant human efforts are required for implementation and encapsulation, enabling agent interaction. Therefore, despite the abundance of software and games available for human use, only a limited number are accessible to agents, especially for commercial closed-source games and software applications. Additionally, the lack of consensus on environment standards further complicates the interaction, as each environment has specific observation and action spaces, tailored to its unique requirements. This variation exacerbates the challenge of enabling agents to interact with diverse environments and collect data with a consistent level of fine-grained semantics to improve the agent’s capabilities. Few agents can complete tasks across multiple environments so far.

Similar to OpenAI Universe (OpenAI, 2016) and SIMA (Raad et al., 2024), our goal is to explore a unified way that allows agents to interact for measuring and training agents’ abilities across a wide range of games, websites, and other applications without heavy human efforts needed. This approach aims to prove that diverse software applications and games can serve as out-of-the-box environments for AI development.

B.2 LMM-BASED AGENTS FOR COMPUTER TASKS

Agents for Software Manipulation. Agents for software applications are developed to complete tasks such as web navigation (Zhou et al., 2023; Deng et al., 2023; Mialon et al., 2023) and software application control (Rawles et al., 2023; Yang et al., 2023b; Kapoor et al., 2024). While previous LLM-based web agents (Deng et al., 2023; Zhou et al., 2023; Gur et al., 2023; Zheng et al., 2024b) show some promising results in effectively interacting with content on webpages, they usually use raw HTML code and DOM tree as input and interact with the available element IDs, ignoring the rich visual patterns with key information, like icons, images, and spatial relations. Recently, multimodal web agents (Yan et al., 2023; Gao et al., 2023; He et al., 2024; Zheng et al., 2024a; Niu et al., 2024; Zhang et al., 2024; Wu et al., 2024) and mobile app agents (Yang et al., 2023b; Wang et al., 2024b) have been explored. Though using screenshots as input, they still rely on built-in APIs and advanced tools to get internal information, like available interactive element IDs, to execute corresponding actions, which greatly limits their applicability. Other train-based agents (Hong et al., 2023; Furuta et al., 2023; Cheng et al., 2024) also suffer from generalizing to unseen software and tasks. Moreover, all of these works primarily focus on static websites and software, which greatly reduces the need for timeliness and simplifies the setting by ignoring the dynamics between adjacent screenshots, *i.e.*, animations, and incomplete action space without considering the duration of the key press and different mouse mode. It results in the failure of deployment to the tasks with rapid graphics changes, *e.g.*, game playing.

Agents for Game Playing. Several attempts try to develop foundation agents for complex video games, such as Minecraft (Wang et al., 2023b;a; 2024a), Starcraft II (Ma et al., 2023) and Civilization-like game (Qi et al., 2024) with textual observations obtained from internal APIs and pre-defined semantic actions. Although JARVIS-1 (Wang et al., 2023a) claims to interact with the environment in a human-like manner with the screenshots as input and mouse and keyboard for control, its action space is predefined as a hybrid space composed of keyboard, mouse, and API. The game-specific observation and action spaces prohibit the generalization of them to other novel games. Pre-trained with videos with action labels, VPT (Baker et al., 2022) manages to output mouse

and keyboard control with raw screenshots as input without any additional information. However, collecting videos with action labels is time-consuming and costly, which is difficult to generalize to multiple environments. Another concurrent work, SIMA (Raad et al., 2024) trained embodied agents to complete 10-second-long tasks over ten 3D video games. Though their results are promising to scale up, they focus on behavior cloning with gameplay data from human experts, resulting in a high expense.

In both targeting complex video games and diverse software applications, **CRADLE** attempts to explore a new way to efficiently interact with different complex environments in a unified manner and facilitate further data collection. In a nutshell, to our best knowledge, there are currently no agents under the GCC setting, reported to show superior performance and generalization in complex video games and across computer tasks. In this work, we make a preliminary attempt to explore and benchmark diverse environments in this setting, applying our framework to diverse challenging environments under GCC and proposing an approach where any software can be used to benchmark agentic capabilities in it.

C EXPERIMENTAL COST

Table 6: Financial and time-related costs of running all the tasks once in each environment or domain.

	RDR2	Cities: Skylines	Stardew Valley	Dealer’s Life 2	Software Apps	OSWorld	Total
Tasks Num.	14	1	3	1	25	369	-
Input Tokens	600M	150M	60M	25M	45M	-	-
Output Toekns	20M	7.5M	4M	1M	2.5M	-	-
Cost (USD)	\$3300	\$862.5	\$345	\$140	\$262.5	\$500	\$5410
Time	240 hrs	60 hrs	30 hrs	20 hrs	50 hrs	240 hrs	640 hrs

Table 6 shows the approximate cost of experiments in Section 4.2 with gpt-4o-2024-05-13. Baselines comparison and ablation studies are not included. Since all the tasks were run 5 times except for OSWorld once, the total cost of getting all the results shown in Section 4.2 is approximately 5400 USD. claude-3-opus-20240229 will roughly use 3X more money and 2X more time compared to gpt-4o-2024-05-13, due to its higher price and longer latency. We also want to note that with the latest model, gpt-4o-2024-08-06, the cost will be halved. We estimate that costs will decrease by one or two orders of magnitude in the coming few years. Then the cost will be affordable to every researcher and developer.

D GENERAL IMPLEMENTATION

Here we introduce the general implementation details of **CRADLE**. For specialized implementations addressing issues unique to their own environment, please refer to the corresponding section.

Hardware. All software and games can be run on regular Windows 10 machines, except for RDR2, which is tested on machines with RTX-4090 GPU separately.

Backbone Model. We employ GPT-4o (OpenAI, 2024b), currently one of the most capable LMM models, as the framework’s backbone model. If not mentioned explicitly, all the experiments are done with gpt-4o-2024-05-13. Temperature is set to 0 to lower the variance of the text generation. Same as Voyager (Wang et al., 2024a), we use OpenAI’s *text-embedding-ada-002 model* (OpenAI, 2022) to generate embeddings for each skill, stored in the procedural memory and retrieved according to the similarities.

Evaluation Methods. Unlike conventional research benchmarks, which usually provide grounding signals for evaluation, it is difficult to have a unified and general method to determine whether a task is completed automatically in diverse software, especially in video games. Similarly to SIMA (Raad et al., 2024), we apply human evaluation to all tasks across application software and games. Moreover, to provide more quantitative results and a comparison baseline, we provide results for the OSWorld (Xie et al., 2024) benchmark, a contemporaneous benchmark that provides evaluation scripts for at least one solution per task.

Observation Space. CRADLE only takes a video clip, which records the progress of execution of the last action, as input. To lower the frequency of interaction with backbone models and reduce the strain on the computer, video is recorded at 2 fps (a screenshot every 0.5 seconds), which proves to be sufficient in most cases for information gathering without missing any important information. It is important to note that, due to the dynamism of the RDR2 and Stardew Valley and the LMM inference and communication latency, we must pause those game environments while waiting for backbone model responses. Other environments execute continuously.

Action Space. For the action space, it includes all possible keyboard and mouse operations, including `key_press`, `key_hold`, `key_release`, `mouse_move`, `mouse_click`, `mouse_hold`, `mouse_release`, and `wheel_scroll`, which can be combined in different ways to form combos and shortcuts, use keys in fast sequence, or coordinate timings. We choose to use Python code to simulate these operations and encapsulate them into an `io_env` class. Skill code needs to be generated by the agent in order to utilize such functions and affordances so executed actions take effect. Table 7 illustrates CRADLE’s action space.

Table 7: Action space in the CRADLE framework, including action attributes. Coordinate system is either *absolute* or *relative*. Actions with durations can be either *synchronous* or *asynchronous*.

Type	Action	Attributes
Keyboard	Key Press	Key name (string), Key press duration (seconds:float)
	Key Hold	Key name (string)
	Key Release	Key name (string)
	Key Combo	Key names (strings), Key combo duration (seconds:float), Wait behaviour (sync/async)
	Hotkey	Key names (strings), Hotkey sequence duration (seconds:float), Wait behaviour (sync/async)
	Text Type	String to type (string), Typing duration (seconds:float)
Mouse	Button Click	Mouse button (left/middle/right), Button click duration (seconds:float)
	Button Hold	Mouse button (left/middle/right)
	Button Release	Mouse button (left/middle/right)
	Move	Mouse position (width:int, height:int), Mouse speed (seconds:float), Coordinate system (relative/absolute), Tween mode (enum) ²
	Scroll	Orientation (vertical), Distance (pixels:int), Duration (seconds:float)
Wait	Noop	-

It is important to note that, while some works (e.g., AssistantGUI (Gao et al., 2023), OmniACT (Kapoor et al., 2024) and OSWorld (Xie et al., 2024)) use *PyAutoGUI*³ for keyboard and mouse control, this approach does not work in all applications, particularly in modern video games using DirectX⁴. Moreover, such work chooses to expose a subset of the library functionality in its action space, ignoring dimensions like press duration and movement speed, which are critical in many scenarios (e.g., RDR2, for opening the weapon wheel and changing view).

³Python library that provides a cross-platform GUI automation module - <https://github.com/asweigart/pyautogui>

⁴Microsoft DirectX graphics provides a set of APIs for high-performance multimedia apps - <https://learn.microsoft.com/en-us/windows/win32/directx>

To ensure wide game and software compatibility and accommodate different operating systems, in our current implementation we use the similar *PyDirectInput* library⁵ and *PyAutoGUI* for keyboard control, utilize *AHK*⁶ and write our own abstraction (using the *ctypes* library⁷) to send low-level mouse commands to the operating system for mouse control. For increased portability and ease of maintenance, all keyboard and mouse control is encapsulated in a class, called *IO_env*.

Notably, our low-level control wrapper is adapted for both MacOS and Windows systems, making the OS transparent to us. At the software window level, we implemented automatic switching between the target software window and the window running the agent (using Python *ctypes* for Windows and *AppleScript* for MacOS⁸).

Procedure Memory. This memory stores pre-defined basic skills and the generated skills captured from the *Skill Curation*. However, as we continuously obtain new skills during game playing, the number of skills in procedural memory keeps increasing, and it is hard for GPT-4o to precisely select the most suitable skill from the large memory. Thus, similar to Voyager (Wang et al., 2024a), we use OpenAI’s *text-embedding-ada-002 model* (OpenAI, 2022) to generate embeddings for each skill and store pre-defined basic skills and any generated skills captured from *Skill Curation*, along with their embeddings in a procedural memory. We retrieve a subset of skills, that are relevant to the given task, and then let GPT-4o select the most suitable one from the subset. In the skill retrieval, we pre-compute the embeddings of the documentations (code, comments and descriptions) of skill functions, which describe the skill functionality, and compute the embedding of the given task. Then we compute the cosine similarities between the skill documentation embeddings and the task embedding. The higher similarity means that the skill’s functionality is more relevant to the given task. We select the top K skills with the highest similarities as the subset. Using similarity matching to select a small candidate set simplifies the process of choosing skills.

Episodic Memory. This memory stores all the useful information provided by the environment and LMM, which consists of short-term memory and long-term summary.

The short-term memory stores the screenshots within the recent k interactions in game playing and the corresponding information from other modules, e.g., screenshot descriptions, task guidance, actions, and reasoning. We set k to five, and it can be regarded as the memory length. Information stored over k interactions ago will be forgotten from direct short-term memory. Empirically, we found that recent information is crucial for decision-making, while a too-long memory length would cause hallucinations. In addition, other modules continuously retrieve recent information from short-term memory and update the short-term memory by storing the newest information.

For some long-horizon tasks, short-term memory is not enough. This is because the completion of a long-horizon task might require historical information from a long steps ago. For example, the agent might do a series of short-horizon tasks during a long-horizon task, which makes the original long-horizon task forgotten in short-term memory. To maintain the long-term valuable information while avoiding the long-token burden of GPT-4o, we propose a recurrent information summary as long-term memory, which is the text summarization of experiences in game playing, including the ongoing task, the past entities that the player met, and the past behaviors of the player and NPCs.

In more detail, we provide GPT-4o with the summarization before the current screenshot and the recent screenshots with corresponding descriptions, and GPT-4o will make a new summarization by organizing the tasks, entities, and behaviors in the time order with sentence number restriction. Then we update the summarization to be the newly generated one, which includes the information in the current screenshot. The recurrent summarization update, inspired by RNN, achieves linear-time inference by preserving a hidden state that encapsulates historical input. This method ensures

⁵Python library encapsulating Microsoft’s *DirectInput* calls for convenience manipulating keyboard keys - <https://github.com/learncodebygaming/pydirectinput>

⁶A fully typed Python wrapper around AutoHotkey to keyboard and mouse control - <https://github.com/spyoungtech/ahk>

⁷Python library that provides C compatible data types, and allows calling functions in DLL/.so binaries - <https://docs.python.org/3/library/ctypes.html>

⁸AppleScript is a scripting language created by Apple, which allows users to directly control scriptable applications, as well as parts of MacOS - https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptLangGuide/introduction/ASLR_intro.html

the compactness of summarization token lengths and recent input data. Furthermore, the incorporation of long-term memory enables the agent to effectively retain crucial information over extended periods, thereby enhancing decision-making capabilities.

Information Gathering. Given the video clip as input, we mainly depend on GPT-4o’s OCR capabilities to extract textual information in the keyframes, which usually contain critical guidance and notifications for the current situation. We also rely on GPT-4o’s visual understanding to analyze the visual information in the frames. Besides, we augment LMMs’ visual understanding via some tools, like template matching (Brunelli, 2009), Grounding DINO (Liu et al., 2023), and SAM (Kirillov et al., 2023), to provide additional grounding for object detection and segmentation. Some visual prompting tricks, like drawing axes and colorful directional bands, are also applied to enhance the GPT-4o’s visual ability.

Task Inference. After reflecting on the outcome of the last executed action, We let GPT-4o analyze the current situation to infer the most suitable task for the current moment and estimate the highest priority task to perform and when to stop an ongoing task and start a new one.

Skill Curation. GPT-4o is required to strictly follow the provided interfaces and examples to generate the corresponding code for new skills. Moreover, GPT-4o is required to include documentation/comments within the generated code, delineating the functionality of each skill. *Procedural Memory* where skills are stored will then check whether the code is valid, whether the format of documentation is right, and whether any skill with the same name already exists. If all conditions are passed, the newly generated skill is persisted for future utilization.

Action Planning. GPT-4o needs to select the appropriate skills from the curated skill set and instantiate these skills into a sequence of executable actions by specifying any necessary parametric aspects (e.g., duration, position, and target) according to the current task and history information. The generated action is then fed to the *Executor* for interaction with the environment.

E RED DEAD REDEMPTION II

E.1 INTRODUCTION TO RDR2

Red Dead Redemption II (RDR2) is an epic AAA Western-themed action-adventure game by Rockstar Games. As one of the most famous and highest-selling games in the world, it is widely acknowledged for its movie-like realistic scenes, rich storylines, and immersive open-ended world. The game applies a typical role-playing game (RPG) control system, played from a first- or third-person perspective, which uses WASD for movement, mouse control for view changing, first- or third-person shooting for combat, and inventory and manipulation.

For most of the game, players need to control the main character, Arthur Morgan, upon choosing to complete mission scenarios following the main storyline. Otherwise, they can freely explore the interactive world, such as going hunting, fishing, chatting with non-player characters (NPCs), training horses, witnessing or partaking in random events, and participating in side quests. As the main storyline progresses, different skills are gradually unlocked. As a close-source commercial game, no APIs are available for obtaining additional game-internal information nor pre-defined automation actions. Following its characteristics, this game serves as a fitting and challenging environment for the GCC setting and a comprehensive benchmark for embodiment.

E.2 OBJECTIVES

In Chapter 1 of RDR2, the first two missions of the main storyline are *Outlaws from the West* and *Enter, Pursued by a Memory*. These missions serve as the tutorial content for RDR2, guiding players step-by-step into the role of Arthur. They immerse the player in the story’s development while teaching the game’s controls and mechanics.

We divided Mission 1 and Mission 2 into 8 and 5 tasks respectively based on the checkpoints within each mission. Each checkpoint may present failure scenarios. For example, in Mission 1, there are six failure scenarios: i) Assaults, kills, or abandons Dutch or Micah; ii) Allows Dutch or Micah to be killed; iii) Abandons the homestead; iv) Assaults, kills, or abandons their horse; v) Assaults, kills,

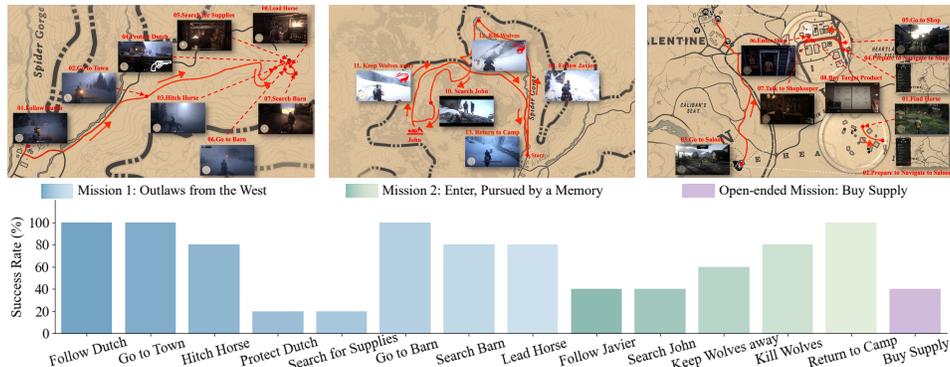


Figure 10: Trajectory and success rates of 13 main storyline tasks and 1 open-world task in RDR2. Each task is run 5 times and each trial is run for at most 500 steps. Long horizontal and challenging tasks like *Protect Dutch* and *Search for Supplies* usually need several times of retry to complete, resulting in the demand for more steps. It explains the low success rate of these tasks within 500 steps.

or abandons the horse in the barn; vi) Dies. We categorized each sub-task as either "Easy" or "Hard" based on the likelihood of failure at each checkpoint and the need to retry the checkpoint.

To evaluate **CRADLE**'s capabilities in an open-world environment, Mission 3 is designed as a hard open-ended task. Unlike the first two tutorial missions, it does not include any checkpoints. Consequently, the entire Mission 3 is treated as a single, comprehensive task. Although we do not subdivide Mission 3 into finer tasks, we aim to identify key points to facilitate a clearer understanding of Mission 3 for the reader.

Tables 8 and 9 provide a brief introduction of each task in the first two missions of the main storyline and an open-ended mission, along with approximate estimates of their difficulty. Due to GPT-4o's poor performance in spatial understanding and fine-manipulation skills, it can be challenging for our agent to perform certain actions, like entering or leaving a building, or going to precise indoor locations to retrieve specific items. Additionally, the high latency of GPT-4o's responses also makes it harder for an agent to deal with time-sensitive events, *e.g.*, during combat.

E.3 IMPLEMENTATION DETAILS

Our experiments are based on the latest version of RDR2, 'Build 1491.50'. As shown in Figure 14, strictly following the GCC setting, our agent takes the video of the screen as input and outputs keyboard and mouse operations to interact with the computer and the game. An observation thread is responsible for the collection of video frames from the screen and each video clip records the whole in-game process since executing the last action.

Information Gathering. To extract keyframes from the video observation, we utilize the VideoSubFinder tool⁹, a professional subtitle discovery and extraction tool. These keyframes usually contain rich meaningful textual information in the game, which are highly relevant to the completion of tasks and missions (such as character status, location, dialogues, in-game prompts and tips, etc.) We use GPT-4o to extract and categorize all the meaningful contexts in these keyframes and perform OCR, and call this processing "gathering text information". Then, to save interactions with GPT-4o, we only let GPT-4o provide a detailed description of the last frame of the video.

While GPT-4o exhibits impressive visual understanding abilities across various CV tasks, we find that it struggles with spatial reasoning and recognizing some game-specific icons. To address these limitations, we add a visual augmentation sub-module within our *Information Gathering* module. This augmentation step serves two main purposes: i) utilize Grounding DINO (Liu et al., 2023), an open-set object detector, to output precise bounding boxes of possible targets in an image and serve as spatial clues for GPT-4o; and ii) perform template matching (Brunelli, 2009) to provide icon recognition grounding truth for GPT-4o when interpreting instructions or menus shown on screen. As LMM capabilities mature, it should be possible to disable such augmentation.

⁹VideoSubFinder standalone tool - <https://sourceforge.net/projects/videosubfinder/>

1296 Table 8: Tasks in the first two missions of RDR2. In the tutorial guide, the prompt text *Start Dialogue* signifies
 1297 the end of the previous checkpoint and the beginning of the current checkpoint. *Difficulty* refers to how hard to
 1298 accomplish the corresponding tasks. Figures 11 and 12 showcase snapshots of each task (specific sub-figures
 1299 marked in parenthesis in the table). The maximal number of steps (agent takes one action) for each task is 500.

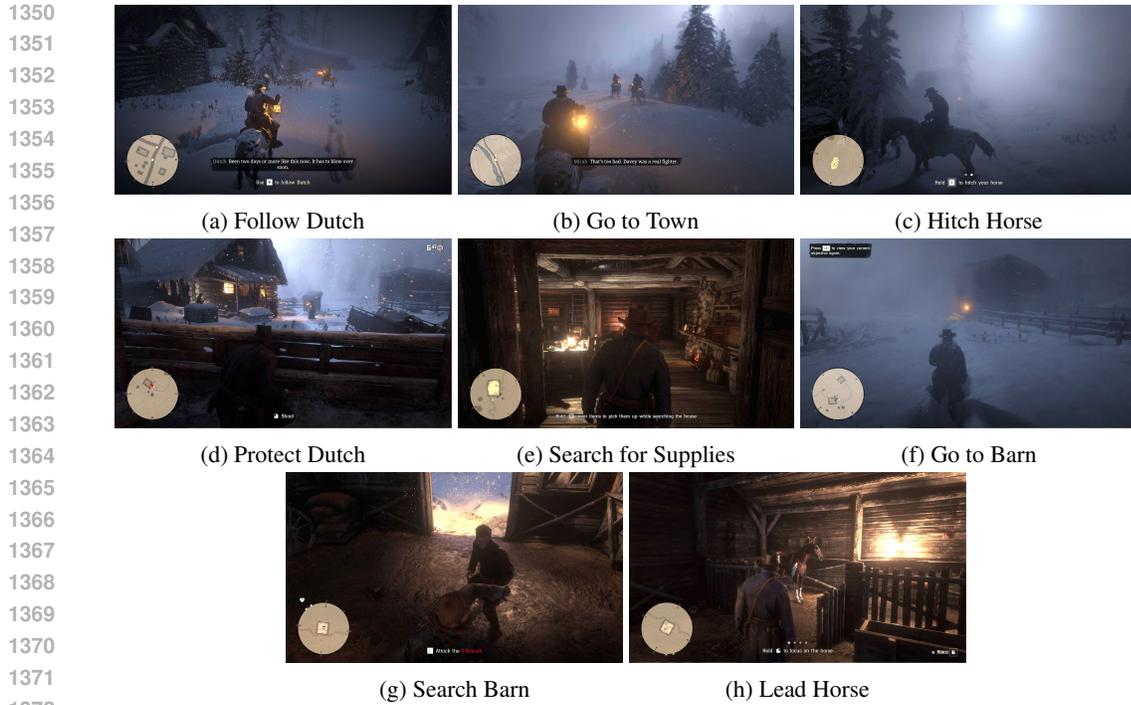
Mission 1: Outlaws from the West	Description	Start Dialogue	Difficulty
1302 Follow Dutch (Fig. 11a)	Arthur follows Dutch on horseback into the snow to find their scouting gang members.	Use [W] to Follow Dutch	Easy
1303 Go to Town (Fig. 11b)	Arthur rides his horse, following Micah to the vicinity of a little homestead Micah discovered.	Hold [W] to match speed with Dutch and Micah	Easy
1304 Hitch Horse (Fig. 11c)	Arthur hitches the horse to the hitching post, then goes to the old shed and takes cover.	Hold [E] to hitch your horse	Easy
1305 Protect Dutch (Fig. 11d)	Arthur uses his gun to shoot all of the O’Driscolls inhabiting the house and protect Dutch.	Use [W] to peak out of cover	Hard
1306 Search for Supplies (Fig. 11e)	Arthur follows Dutch to the house to search for supplies.	Hold [R] near items to pick the up while searching house.	Hard
1307 Go to Barn (Fig. 11f)	Arthur follows Dutch’s directions and goes to the barn to see if there’s anything inside.	Dutch: Micah, Arthur, keep looking for stuff	Easy
1308 Search Barn (Fig. 11g)	Arthur searches the barn and defeats the O’Driscoll hiding inside.	[F] Attack the O’Driscoll	Hard
1309 Lead Horse (Fig. 11h)	Arthur calms the horse and takes it out of the barn.	Hold [Right Mouse Button] to focus on the horse	Easy
Mission 2: Enter, Pursued by a Memory	Description	Start Dialogue	Difficulty
1314 Follow Javier (Fig. 12a)	Arthur rides his horse following Javier up the mountain through the blizzard searching for John’s trail.	Follow Javier	Hard
1315 Search John (Fig. 12b)	After dismounting, Arthur followed Javier over slopes and ledges to find John and carry him away.	Javier: Down this way	Hard
1316 Keep Wolves away (Fig. 12c)	Arthur manages to shoot all of the wolves before they can attack Javier and John.	Keep the wolves away from Javier and John	Hard
1317 Kill Wolves (Fig. 12d)	Three people ride horses down the mountain. Arthur eliminate the wolves, protecting Javier and John ahead.	Javier: Come on, let’s get back to the others	Hard
1318 Return to Camp (Fig. 12e)	Arthur followed Javier on horseback back to camp.	Yea... c’mon. Let’s push hard and get back	Easy

1324 Table 9: Key points in the open-ended mission, *Buy Supply* in RDR2. Figure 13 showcases snapshots of key
 1325 points (specific sub-figures marked in parenthesis in the table).

Mission 3: Buy Supply	Description
1327 Find Horse (Fig. 13a)	Find and mount the horse in the camp.
1328 Prepare to Navigate to Saloon (Fig. 13b)	Open map, find the saloon and create waypoint.
1329 Go to Saloon (Fig. 13c)	Ride horse to the saloon.
1330 Prepare to Navigate to Shop (Fig. 13d)	Open map, find the general store and create waypoint.
1331 Go to Shop (Fig. 13e)	Ride horse to the shop.
1332 Enter Shop (Fig. 13f)	Dismount the horse and enter the shop.
1333 Talk to Shopkeeper (Fig. 13g)	Approach the shopkeeper and talk.
1334 Buy Target Product (Fig. 13h)	Open the menu, find and buy the target product.

1337 **Self-Reflection.** The reflection module mainly serves to evaluate whether the previously executed
 1338 action was successfully carried out and whether the current executing task is finished. To achieve
 1339 this, we uniformly sample at most 8 sequential frames from the video observation since the execution
 1340 of the last action and use GPT-4o to estimate the success of its execution. Additionally, we expect
 1341 GPT-4o can also provide analysis for any failure of the last action (e.g., the move-forward action
 1342 failed and the cause could be the agent was blocked by an obstacle). With such valuable information
 1343 as input for *Action Planning*, including the failure/success of the last action and the corresponding
 1344 analysis, the agent is capable of attempting to remedy an inappropriate decision or action execution.

1345 Moreover, some actions require prolonged durations, such as holding down specific keys, which can
 1346 coexist or interfere with other actions decided by subsequent decisions. Consequently, the reflection
 1347 module must also decide whether an ongoing action should continue to be executed. Furthermore,
 1348 self-reflection can be leveraged to dissect why the last action failed to bring the agent close to the
 1349 target task completion, better understand the factors that led to the successful completion of the
 preceding task, and so on.



1373 Figure 11: Image examples of tasks in the first mission of *Outlaws from the West*. (The picture has been
1374 brightened for easier reading.)



1391 Figure 12: Image examples of tasks in the second mission of *Enter, Pursued by a Memory*.

1394 Besides, we observe that instead of providing GPT-4o with sequential high-resolution images for
1395 self-reflection, low-resolution images make it easier for GPT-4o to understand the relation among
1396 the sequential screenshots and capture dynamic changes, resulting in a significantly higher success
1397 rate of detecting whether the action is executed successfully and take any effect. We hypothesize that
1398 since a high-resolution image can cost as many as 2000 tokens, too many high-resolution images
1399 make GPT-4o fail to capture the overall changes across screenshots and be caught up in the local
1400 details.

1401 **Task Inference.** During gameplay, we let GPT-4o propose the current task to perform whenever it
1402 believes it is time to start a new task. GPT-4o also outputs whether the task is a long- or short-horizon
1403 task when proposing a new task. Long-horizon tasks, such as traveling to a location, typically require
multiple iterations, whereas short-horizon tasks, like picking up an item or conversing with someone,

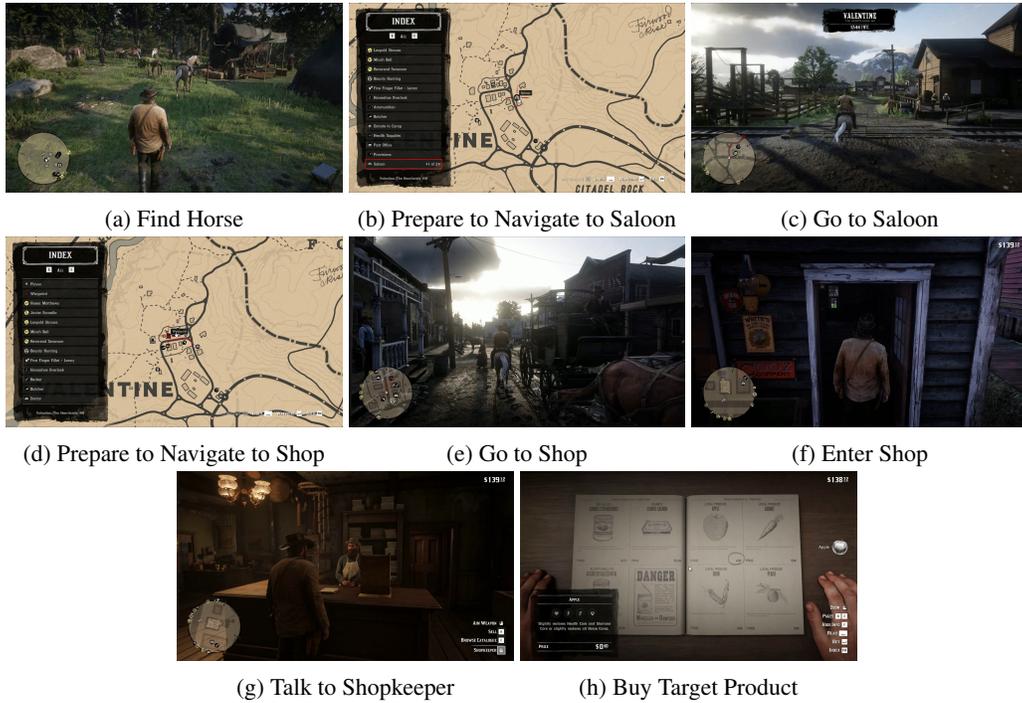


Figure 13: Image examples of key points in the open-ended task of *Buy Supply*.

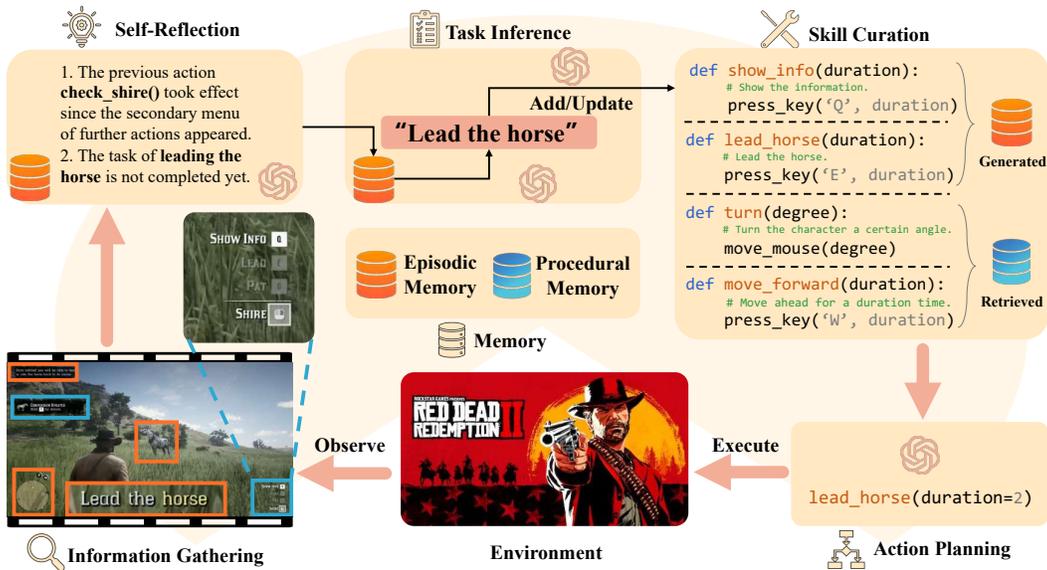


Figure 14: The detailed illustration of how **CRADLE** is instantiated as a game agent to play RDR2.

involve fewer iterations. The agent will follow the newly generated task for the next 3 interactions. After 3 interactions, the agent returns to the last long-horizon task in the stack. Deciding on a binary task horizon is much easier and more robust for GPT-4o, than re-planning at every iteration. Since a long-horizon task frequently includes multiple short-horizon sub-tasks, this implementation also helps avoid forgetting the long-horizon tasks under execution.

Skill Curation. As shown in Figure 16, during gameplay, instructions often appear on the screen, such as "press [Q] to take over" and "hold [TAB] to view your stored weapons", which serve as essential directives for completing current and future tasks proficiently. To save interactions with

GPT-4o, we implement a simple version of this module inside *Information Gathering* to reduce interactions with GPT-4o. When GPT-4o detects and classifies some instructional text in the recent observation, which usually contains key and button hints, it will directly generate the corresponding code and description.

Action Planning. Upon execution of this module, we first retrieve the top k relevant skills for the task from procedural memory, alongside the newly generated skills. We then provide GPT-4o with the current task, the set of retrieved skills, and other information collected in *Information Gathering* that may be helpful for decision-making (e.g., recent screenshots with corresponding descriptions, previous decisions, and examples) and let it suggest which skills should be executed. We also request that GPT-4o provide the reasons for choosing these skills, which increases the accuracy, stability, and explainability of skill selection and thus greatly improves framework performance. While GPT-4o sometimes may generate a sequence of actions, we currently only execute the first one, and perform *Self-Reflection*, since we observe a tendency for the second action to usually suffer from severe hallucinations.

Action Execution. Unlike the conventional mouse operation in standard software, where the cursor is restricted to a 2D grid and remains visible on the screen to navigate and interact with elements, the utilization of the mouse in 3D games like RDR2 introduces a varied control scheme. In menu screens, the mouse behaves traditionally, offering familiar point-and-click functionality. However, during gameplay, the mouse cursor disappears, requiring players to move the mouse according to specific action semantics. For example, to alter the character’s viewpoint, the player needs to map the actual mouse movement to in-game direction angle changes, which differ in magnitude in the X and Y axes. Another special transition applies to shooting mode, where the front sight is fixed at the center of the screen, and players must maneuver the mouse to align the sight with target enemies. This nuanced approach to mouse control in different contexts adds an extra layer of challenge to general computer handling, showcasing the adaptability required in game environments, compared to regular software applications.

Procedural Memory. In our target setting, We intend to let the agent learn all skills from scratch, to the extent possible for the main storyline missions. The procedural memory is initialized with only preliminary skills for basic movement, which are not clearly provided by the in-game tutorial and guidance.

- *turn(degree), move_forward(duration)*: Since the game does not precisely introduce how to move in the world through in-game instructions, we provide these two basic actions in advance, so GPT-4o can perform basic mobility, while greatly reducing the number of calls to the model.
- *shoot(x, y)*: RDR2 also does not provide detailed instructions on how to aim and shoot. Moreover, due to limitations with GPT-4o spatial reasoning and the need to sometimes augment images with object bounding boxes, we provide such basic skill for the agent to complete relevant tasks.
- *select_item_at(x, y)*: Similarly to *shoot()*, due to the lack of instructions, we provide such skill for the agent to move the mouse to a certain place to select a given item.

Beyond these basic atomic low-level actions, we introduce a few composite skills to facilitate the game playing progress. The agent should be able to complete tasks using only the basic skills above and the skills it learns, but these composite skills streamline the process by greatly reducing calls to the backend model.

- *turn_and_move_forward(degree, duration)*: This skill is just a simple composition of *turn()* and *move_forward()* to save frequent calls to GPT-4o in a common sequence.
- *follow(duration)* and *navigate_path(duration)*: In RDR2, tasks often guide players to follow NPCs or generated paths (red lines) in the minimap to certain locations. This can be reliably accomplished via the basic movement skills, but requires numerous interactions with GPT-4o. To control both cost and time budgets involving GPT-4o’s responses, we leverage the information shown in the minimap to implement a composite skill to follow target NPCs or red lines for a short set of game iterations. The default duration is 20 iterations. Increasing the duration can dramatically improve the performance in task *Follow Dutch*, *Follow Javier* and *Killing Wolves* but significantly decrease the success rate of

1512 *Search John* since this task requires frequent exchange of the skills between climbing and
 1513 following.

- 1514 • *fight()*: As output of an interaction with GPT-4o, the agent will only take one action per
 1515 step. However, though the action is generated correctly, specifically in fight scenarios, the
 1516 action frequency may not be high enough to defeat an opponent. In order to allow sub-
 1517 second punches, we provide a pre-defined action that wraps this multi-action punching,
 1518 which can be selected by GPT-4o to effectively win fights.
 1519

1520 For the open-ended mission, since the agent skips all the tutorials in Chapter I, we provide all the
 1521 necessary skills in the procedural memory at the beginning of the mission.

1522 **Episodic Memory.** This module stores all the useful information, *e.g.*, input and output of GPT-
 1523 4o. In each iteration, after the self-reflection, we will request GPT-4o to summary the event that
 1524 happened in the last action and the past experiences.

1525 **Game Pause.** To prevent in-game time from passing in real-time games like RDR2, we have to
 1526 pause the game while waiting for LMMs’ response. The time interval between two consecutive
 1527 actions can be as long as one minute. In RDR2, after the agent finishes executing outputted actions,
 1528 *esc* will be automatically pressed to pause the game and when the agent determines the next action,
 1529 *esc* will be automatically pressed again to unpause the game. Note that there will be an animation
 1530 lasting up to 0.5 seconds for both pausing and unpausing. During this animation, we can not control
 1531 the character, but the dynamics of the game world keep changing, *e.g.*, the wolves are still moving.
 1532 It introduces additional challenges for the tasks that require precise timing, like combat.
 1533

1534 E.4 CASE STUDIES

1535
 1536 Here we present a few game-specific case studies for more in-depth discussion of the framework
 1537 capabilities and the challenges of the GCC setting.
 1538

1539 E.4.1 SELF-REFLECTION

1540
 1541 Self-reflection is an essential component in **CRADLE** as it allows our framework reasoning to correct
 1542 previous mistakes or address ineffective actions taken in-game. Figure 15 provides an example of
 1543 the self-reflection module. The task requires the agent to select a weapon to equip, in the context
 1544 of the “Protect Dutch” task. Initially, the agent selects a knife as its weapon by chance, but since
 1545 the game requires a gun to be chosen, this is incorrect and the game still prompts the player to re-
 1546 open the weapon wheel. The self-reflection module is able to determine that the previous action was
 1547 incorrect and on a subsequent iteration the agent successfully opts for the gun, correctly fulfilling
 1548 the task requirement and advancing to the next stage in the story.

1549 E.4.2 SKILL CURATION

1550
 1551 For skill curation, we first provide GPT-4o with examples of general mouse and keyboard control
 1552 APIs, *e.g.*, `io_env.key_press` and `io_env.mouse_click`. Figure 16 shows that GPT-4o can capture
 1553 and understand the prompts appearing on screenshots, *i.e.*, icons and text, and strictly follow the
 1554 provided skill examples using our IO interface to generate correct skill code. Moreover, GPT-4o
 1555 also generates comments in the code to demonstrate the functionality of this skill, which are essential
 1556 for computing similarity and relevance with a given task during skill retrieval. The quality of the
 1557 generated comment directly determines the results of skill retrieval, and further impacts reasoning
 1558 to action planning. Curation can also re-generate code for a given skill, which is useful if GPT-4o
 1559 wrongly recognized a key or mouse button in a previous iteration.

1560 E.4.3 ACTION EXECUTION AND FEEDBACK

1561
 1562 Proper reasoning about environment feedback is critical due to the generality of the GCC setting
 1563 and the level of abstraction to interact with the complex game world. The semantic gaps between
 1564 the execution of an action, its effects in the game world, and observing the relevant outcomes for
 1565 further reasoning lead to several potential issues that **CRADLE** needs to deal with. Such issues can
 be categorized into four major cases:



Figure 15: Case study of self-reflection on re-trying a failed task. Task instruction and context require the agent to equip the gun. A wrong weapon (knife) is first selected, but the agent equips the gun after self-reflection. Only relevant modules are shown for better readability, though all modules (Figure 3) are executed per iteration.



1630 Figure 16: Skill code generation based on in-game instructions. As the storyline progresses, the game will
1631 continually provide prompts on how to use a new skill via keystrokes or utilizing the mouse.

1632
1633
1634

1635 **Lack of grounding feedback.** In many situations, due to the lack of precise information from the
1636 environment, it can be difficult for the system to deduce the applicability or outcome of a given
1637 action. For example, when picking an item from the floor, the action may fail due to the distance to
1638 the object not yet being close enough. Or, if within pick up range, the chosen action may not exactly
1639 apply due to other factors (*e.g.*, character’s package is full).

1640
1641
1642
1643

Even if the right action is selected and executed successfully, the agent still needs to figure out its
results from the partial visual observation of the game world. If the agent needs to pick or manipulate
an object that is occluded from view, the action may execute correctly, but no outcome can be seen.

1644
1645
1646
1647

A representative example in RDR2 happens when the agent tries to pick up its gun from the floor
after a fight. Getting to the right distance, without completely occluding the object, can lead to
multiple re-trials. Figure 17a showcases a situation where, though the character is already standing
near the gun (as seen in the minimap), it’s still not possible to pick it up.

1648
1649
1650

Previous efforts (Wang et al., 2023b; 2024a) that utilize in-game state APIs unreasonably bypass
such issues by leveraging internal structured information from the game and the full semantics of
responses (data) or failures (error messages).

1651
1652
1653
1654
1655
1656
1657
1658
1659

Imprecise timing in IO-level calls. This issue is caused by the ambiguity in the game instructions
or differences in specific in-game action behaviors, where even the execution of a correct action may
fail due to minor timing mismatches. For example, when executing an action like ‘open cabinet’,
which requires pressing the [R] key on the keyboard, if the press is too fast, no effect happens in the
game world. However, as there is no visual change in the game nor other forms of feedback, it can
be difficult for GPT-4o to figure out if an inappropriate action was chosen at this game state or if the
minor timing factor was the problem. Pressing the key for longer triggers an animation around the
button (only if the helper menu is on screen), but this is easily missed and any key release before the
circle completes also results in no effect. Figure 17b illustrates the situation.

1660
1661
1662

The same problem also manifests in other situations in the game, where pressing the same key for
longer triggers a completely different action (*e.g.*, lightly pressing the [Left Alt] key vs. holding it
for longer).

1663
1664
1665
1666
1667

Change in the semantics of key and button. A somewhat similar situation occurs when the same
keyboard key or mouse button gets attributed different semantics in different situations (or even in a
multi-step action). GPT-4o may decide to execute a given skill, but the original semantics no longer
hold. The lack of in-game effect parallels the previous situations. Worse yet, an undesired effect
will confuse the system regarding the correct action being selected or not.

1668
1669
1670
1671
1672
1673

For example, when approaching a farm in the beginning of the game, the agent needs to hitch the
horse to a pole to continue. The operation to perform the action consists of pressing the [E] key near
a hitching post (as shown in Figure 17c). However, the same [E] key press is the only constituting
step in other actions with different semantics, like *dismount the horse* or *open the door*. Wrongly
triggering a horse dismount at the situation shown in the figure can lead to undesired side effects,
i.e., it may mislead the system about the actual effects of the action or affect the planning of which
next actions to perform.

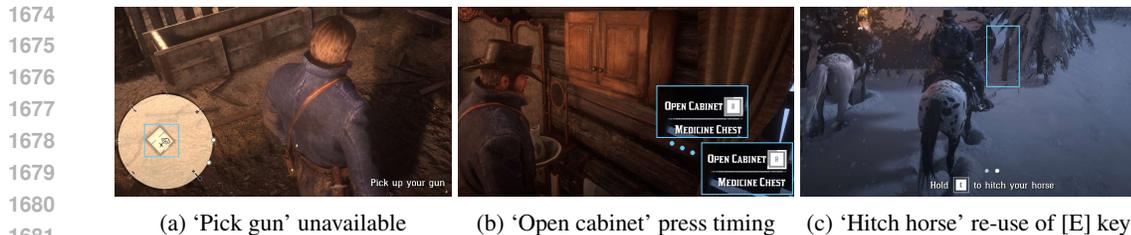


Figure 17: Examples of action execution uncertainty. Lack of environmental feedback to actions and semantic gaps between action intent and game command can lead to challenging situations for agent reasoning.

Interference issues. Lastly, completion of some actions requires the correct execution of multiple steps sequentially, which could be interrupted in many ways not related to the agent’s own actions. Without the use of APIs that expose internal states or other forms of feedback, it is much harder for the agent to decide when to repeat sub-actions or try different strategies. For example, if the agents gets shot and loses aim while in combat, or an unrelated in-game animation is triggered mid-action, canceling it.

Since there is no direct environment feedback, the agent needs to carefully analyze the situation and try to infer if any action step needs re-execution.

E.5 LIMITATIONS OF GPT-4O AND GPT-4V

Deploying **CRADLE** in a complex game like RDR2 requires the backbone LMM model to handle multimodal input, which revealed several limitations of both GPT-4V and GPT-4o, necessitating external tools to enhance overall framework performance. Initial tests and exploration were performed using GPT-4V, as GPT-4o was not yet available. These tests highlighted significant weaknesses in spatial perception, icon understanding, history processing, and world understanding. Upon the release of GPT-4o, further testing demonstrated some notable improvements in spatial perception. However, enhancements in other areas remained marginal, while some regressions were also observed, all indicating the need for additional tools to aid decision-making.

Spatial Perception. As shown in Figure 18a and 19a, GPT-4V’s spatial-visual recognition capability is insufficient for precise fine-grained control, particularly in detecting whether the character is being or going to be blocked and in estimating the accurate relative positions of target objects. In contrast, GPT-4o exhibits a significant enhancement in spatial perception, capable of recognizing obstacles ahead and estimating the approximate relative positions between objects. However, both models require supplementary information, such as bounding boxes of potential target objects, to make fine-grained decisions. These led to the need to augment certain images to provide auxiliary visual clues for decision-making, *i.e.*, bounding boxes of possible target objects.

Icon Understanding. Both GPT-4o and GPT-4V struggle with domain-specific concepts, such as unique icons within the game, which may represent specific targets or refer to certain mouse and key actions. As shown in Figure 18b and 19b, GPT-4V and GPT-4o fail to recognize the left shift, right mouse button, and space icons. Attempts to incorporate few-shot learning to improve image understanding cannot be generalized. Therefore, we match prepared pattern templates, *e.g.*, icon images, against each screenshot to continuously detect and highlight any appearing icons.

Figure 18c and Figure 19c also demonstrate that although GPT-4o performs better than GPT-4V in understanding the mini-map, it still fails to consistently interpret this crucial information regarding the position and direction of the character. This failure in localization leads the agent to sometimes get lost in the town and miss the task target. While the aforementioned issues can be slightly alleviated by providing additional few-shot examples, a significant improvement is only achieved by cropping the image and providing GPT-4o with the exact region containing the icon to be recognized. This dependency on precise input makes the issue challenging and unreliable for decision-making. Although the above issues can be slightly alleviated by providing additional few-shot examples, it can only have an obvious effect if we crop the image and provide the GPT-4o with the region exactly containing the icon to be recognized, which makes the issue intractable.

1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781



ChatGPT
There are **no immediate obstacles** directly in front of the character in the image. The character has a **clear path ahead** on a muddy street in what appears to be a Western-style town setting.

ChatGPT
The shopkeeper in the image is on the **right side**, behind the counter.

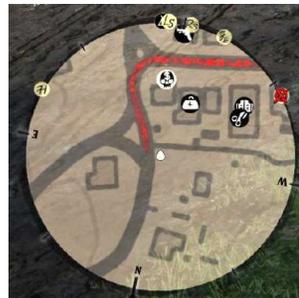
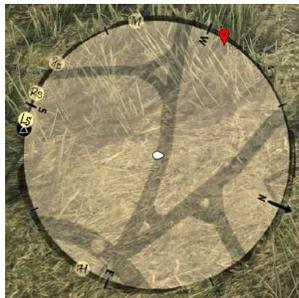
(a) GPT-4V fails to recognize obstacles in the environment and the position of the shopkeeper.



ChatGPT
Based on the provided game screenshot, I've identified the following control prompt:
- "**Hold L**  to ride faster"
This control suggests that holding a certain key (which seems to be an **upward arrow key**, often associated with the **'W' key** for forward movement in many games) will make the character ride faster.

ChatGPT
From the provided image screenshot from the game, I can identify the following keyboard and mouse control prompts:
1. "**Toggle View**" - The key for this action is "**V**".
2. "**Zoom**" - The key for this action is "**O**".
3. "**Flip**" - The key for this action is "**L**".
4. "**Back**" - The key for this action is "**ESC**".

(b) GPT-4V struggles to recognize the icons for keys on keyboard and mouse buttons.



ChatGPT
The central white arrow is facing towards the **left**.

ChatGPT
The central white arrow with the white arrowhead is facing towards the **left**.

(c) GPT-4V cannot understand the correct direction of arrow points, *i.e.*, character, towards in the mini-map.

1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835



ChatGPT
There is **an obstacle** in front of the character. It appears to be a small wooden railing or barrier along the edge of the platform. The character would need to navigate around it or find a way over it to proceed forward.

ChatGPT
The shopkeeper in the image is on the **left** side

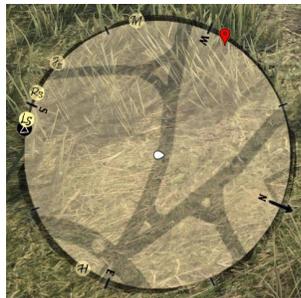
(a) GPT-4o can recognize obstacles in the environment and the position of the shopkeeper.



ChatGPT
Based on the provided image, the control prompt shown is:
• "Hold **L** to ride faster".
The '**L**' refers to the left analog stick on a **controller**, indicating that the player should hold the left analog stick in a forward position to increase the speed of their character's movement, typically while riding a horse.

ChatGPT
From the provided image screenshot from the game, I can identify the following keyboard and mouse control prompts:
1. Toggle View: **V**
2. Zoom: **Mouse Scroll Wheel**
3. Flip: **R**
4. Back: **Esc**
5. Photograph: (No specific key assigned, just a label)

(b) GPT-4o struggles to recognize the icons for keys on keyboard and mouse buttons.



ChatGPT
The central white arrow in the mini-map is facing to the **right**.

ChatGPT
The central white arrow on the mini-map is facing **down**.

(c) GPT-4o occasionally accurately determines the direction of arrow points, *i.e.*, character, towards in the mini-map.

Figure 19: Example situations of GPT-4o's limitations in understanding visual information from the game.

History Processing. Moreover, both GPT-4o and GPT-4V can easily get distracted by irrelevant information in longer contexts, resulting in hallucinations. For example, when action planning utilizes too many historical screenshots, they may confuse past and present frames. Additionally, performance fluctuates and both model versions frequently generate output not adhering to the rules in the provided prompts. To mitigate the issue of hallucinations, we more strictly control input information by further summarizing long-term memory.

World Understanding. Lastly, the absence of an RDR2 world model limits GPT-4V and GPT-4o’s understanding of the consequences of its actions in the game. This often results in inappropriate action selection, such as overestimating the necessary adjustments for aligning targets or misjudging the duration required for certain actions. To alleviate this problem, we introduced extra prompt rules regarding action parameters and more flexibility into the self-reflection module.

F STARDEW VALLEY

F.1 INTRODUCTION TO STARDEW VALLEY

Stardew Valley is an open-ended country-life RPG game developed by ConcernedApe, which has a 98% positive rating on Steam and is rated as Overwhelmingly Positive. Players take on the role of a character disillusioned with city life who inherits a dilapidated farm from their late grandfather. Initially, the farmland is overrun with boulders, trees, stumps, and weeds, which players must clear to make way for crops, buildings, and placeable items. The main goal is to restore and expand the farm through activities such as planting crops, raising animals, mining, fishing, and crafting. Additionally, players can interact with NPCs in town, forming relationships that can lead to marriage and children. Players complete quests for money or to restore the town’s Community Center by completing "bundles," which reward items like seeds and tools and unlock new areas and game mechanics. All activities are balanced against the character’s health, energy, and the game’s clock. Food provides buffs, health, and energy. The game features a simplified calendar with four 28-day months representing each season, affecting crop growth and activities. Compared to RDR2, this game is more lightweight and easy to control. This game features a wealth of production and social activities, presenting a comprehensive test of an agent’s abilities, which is an ideal platform to observe and evaluate agents’ comprehensive behaviors and abilities, like in the Generative Agents (Park et al., 2023). We use the latest version (1.6.8) of the game to conduct all the experiments.

F.2 OBJECTIVES

We find that GPT-4o surprisingly struggles with accurately recognizing and locating objects near the player in this 2D game. This leads to difficulties for the agent to interact with objects or people, as it requires the player to stand precisely in front of them in the grid (*e.g.*, when entering doors, using a pickaxe to break stones). Even some basic tasks are already challenging enough for current agents in this game. Therefore, as shown in Figure 20, we evaluate three essential tasks in the early stages of the game:

- **Farm Cleanup.** Clear the obstacles on the farm, such as weeds, stones, and trees, as much as possible to prepare for farming. This task requires agents to move precisely to be in front of the obstacles, identify the type of obstacles correctly and select corresponding tools to deal with them.
- **Cultivation.** Use the hoe to till the soil, use a parsnip seed packet on the tilled soil to sow a crop, water the crop every day and harvest at least one parsnip. This task requires long-horizontal memory and reasoning.
- **Shopping.** Go to the general store in the town, which is on the other map, to buy more seeds and return home. This task is used to evaluate agents’ long-distance navigation ability.

For each task, the maximal steps is 100.

F.3 IMPLEMENTATION DETAILS

Visual Prompting. As a cartoon-style pixel game, the game screen of Stardew is quite different from the real world. Although GPT-4o can observe coarse-grained information from screenshots,



Figure 20: Three tasks in Stardew Valley.

1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940

more fine-grained information is required to complete tasks. Therefore, as shown in Figure 21, we divide each screenshot into 3×5 grids and require GPT-4o to describe the screenshot in a grid-by-grid format. We empirically find that it can result in a more precise and accurate description. And GPT-4o can also make better control based on the grids. In addition, we also augment the image with two blue and yellow bands on the left and right sides, respectfully, with the prompt, "The blue band represents the left side and the yellow band represents the right side". Our empirical results show that this method significantly improves GPT-4o's ability to accurately distinguish left from right.

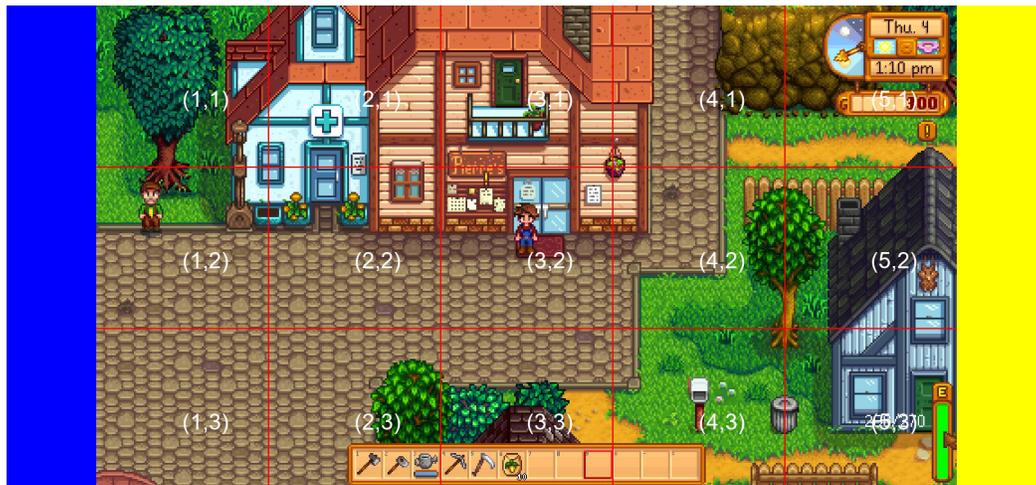


Figure 21: Augmented screenshot via visual prompting. The full screenshot is divided into 3×5 grids and each grid has a unique white coordinate. Additionally, we augment all input images with color bands, with the prompt, "The blue band represents the left side and the yellow band represents the right side", which significantly improves GPT-4o's ability to accurately distinguish left from right.

1944 **Information Gathering.** As mentioned in the introduction of visual prompting, we let GPT-4o
 1945 describe the image grid by grid, which is helpful in locating the position of the character, surrounding
 1946 objects and buildings and facilitates the understanding of the relative positions among them for GPT-
 1947 4o. Besides, while compared to GPT-4V, GPT-4o is able to recognize most of the icons and their
 1948 quality in the toolbar shown at the bottom of the screenshot, GPT-4o cannot output the items in the
 1949 inventory sequentially one by one as it always skips a few in between. We have to clip the box
 1950 for each item out of the toolbar and feed them to GPT-4o independently, augmented with template
 1951 matching, for recognition, which turns out to be more accurate. The success of recognition of the
 1952 tools in the toolbar is critical to tasks like **Farm Cleanup** and **Cultivation**.

1953 **Self-Reflection.** The duration of actions in Stardew is usually much shorter than in RDR2, so we
 1954 only use the first and last frame from the video observation to reduce the number of tokens used
 1955 per request. Additionally, we provide some helpful prior information for GPT-4o. For example, a
 1956 screenshot of the inside of the store is provided to check whether the store was successfully entered.
 1957 This is useful because there are many other buildings near the store, and sometimes GPT-4o controls
 1958 the character to enter the wrong one. However, this is not realized if the screenshot is not provided.

1959 **Skill Curation.** For skill curation, as mentioned in Figure 4, we mainly rely on the in-game manual
 1960 to generate atomic skills, like *move_up()*, *do_action()* and *use_tool()*. In addition, to handle the
 1961 challenges of locating objects, especially doors, we have a special set of composite skills specifically
 1962 for Stardew. *e.g.*, *go_through_door*, *buy_item*, *get_out_of_house* and *enter_door_and_sleep*. With
 1963 the restrictions of GPT-4o in fine-grained control, we designed *go_through_door* composite skills for
 1964 the agent to control the game character to accurately reach various doors and successfully enter, such
 1965 as the house and the store door. and in order to buy certain items such as parsnip seeds, we designed
 1966 the composite skills *buy_item* to control the game character to interact with the salesman and buy
 1967 parsnip seeds. similarly, we designed the *get_out_of_house* and *enter_door_and_sleep* composite
 1968 skills to accurately exit the house from the bed and enter the house and walk to the bed.

1969 **Action Planning.** In this game, we let GPT-4o output at most two skills in a single action every
 1970 time, which turns out to be efficient. The agent usually needs to select the correct tool first and then
 1971 use the tool or do action.

1972 **Procedure Memory.** Procedure Memory is used to store and retrieve skills in code form. In order
 1973 for agents to quickly get started and complete some special tasks in Stardew, we have predefined
 1974 skills in Procedure Memory. These skills are divided into atomic and composite skills. atomic skill
 1975 consists of basic operations such as moving, selecting tools, etc. The description of all the atomic
 1976 skills is listed as follows:

- 1977 • *do_action()*: The function to perform a context-specific action on objects or characters.
- 1978 • *use_tool()*: The function to execute an in-game action commonly assigned to using the
- 1979 character’s current selected tool.
- 1980 • *move_up(duration)*: The function to move the character upward (south) by pressing the ‘w’
- 1981 key for the specified duration.
- 1982 • *move_down(duration)*: The function to move the character downward (north) by pressing
- 1983 the ‘w’ key for the specified duration.
- 1984 • *move_left(duration)*: The function to move the character left (west) by pressing the ‘w’ key
- 1985 for the specified duration.
- 1986 • *move_right(duration)*: The function to move the character right (east) by pressing the ‘w’
- 1987 key for the specified duration.
- 1988 • *select_tool(key)*: The function to select a specific tool from the in-game toolbar based on
- 1989 the given tool number.

1992 and the composite skills are designed for the agent to complete a variety of special tasks. The
 1993 description of all the composite skills is listed as follows:

- 1995 • *buy_item()*: The function to interact with the salesman and buy the item.
- 1996 • *enter_door_and_sleep()*: The function to enter the house and walk to the bed.
- 1997 • *get_out_of_house()*: The function to accurately exit the house from the bed

1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051

- `go_through_door()`: The function to reach and enter all kinds of doors.

Game Pause. The game will pause automatically when the game window is not focused. So when the character finishes executing actions, we will activate another window, *e.g.*, code window, to pause the game and stop the passage of the time in the game.

F.4 CASE STUDIES

Here we present a few game-specific case studies to further discuss **CRADLE**'s self-reflection and task-inference processes in the GCC setting.

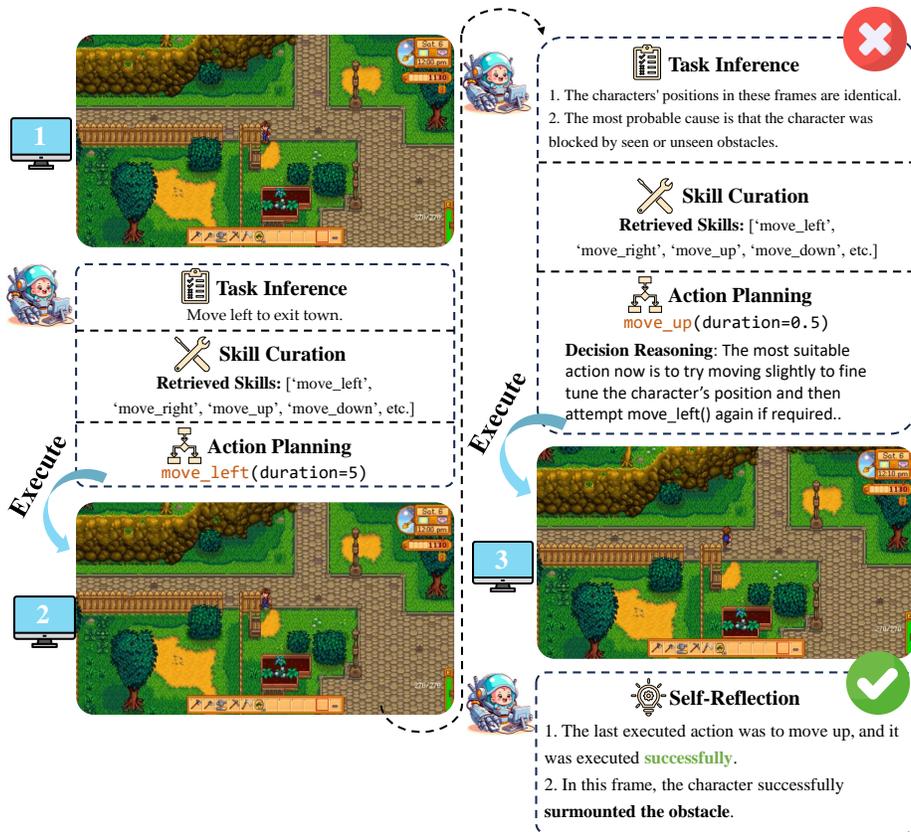


Figure 22: Case study of self-reflection on re-trying a failed task. Task instruction and context require the agent to exit town. A wrong direction is first selected, but the agent moves up after self-reflection. Only relevant modules are shown for better readability, though all modules (Figure 3) are executed per iteration.

F.4.1 SELF-REFLECTION

The Self-reflection module plays an important role in the completion of game missions in Stardew, giving our framework the ability to determine if the actions performed are complete and effective and to correct the errors of invalid actions. In the "Purchasing Seeds" task, the Agent is asked to return home from the store after purchasing items. At the "Home is on the left side of the store" prompt, the Agent controls the character to go left, but there are obstacles to keep going left, and the character must go up to circumnavigate the obstacles. As shown in the Figure 22, the role will initially be stuck at the obstacle and cannot continue to the left. Through Self-Reflection, the Agent can judge that it is currently in a state of obstruction, and moving to the left cannot be implemented smoothly. Therefore, the agent can adjust the direction upward to bypass the obstacle and enable the role to continue to the left until it returns home.

F.4.2 TASK-INFERENCE

Task Inference is a very effective module for completing game quests in Stardew. Its function is to decompose a vague and grand task into a specific sub-task, which effectively guides the Agent to complete the overall task. For example, in the Farming task, as shown in Figure 23, the task that the character needs to complete is "cultivate and harvest a parsnip." This is a complete but vague task. Through the Task Inference module, the Agent breaks down the task into (1) till the soil with the hoe, (2) plant the parsnip seeds, (3) water the planted seeds once daily for four days, (4) harvest the fully grown parsnip. This enables the Agent to know more clearly the steps needed to complete and finish the task successfully.

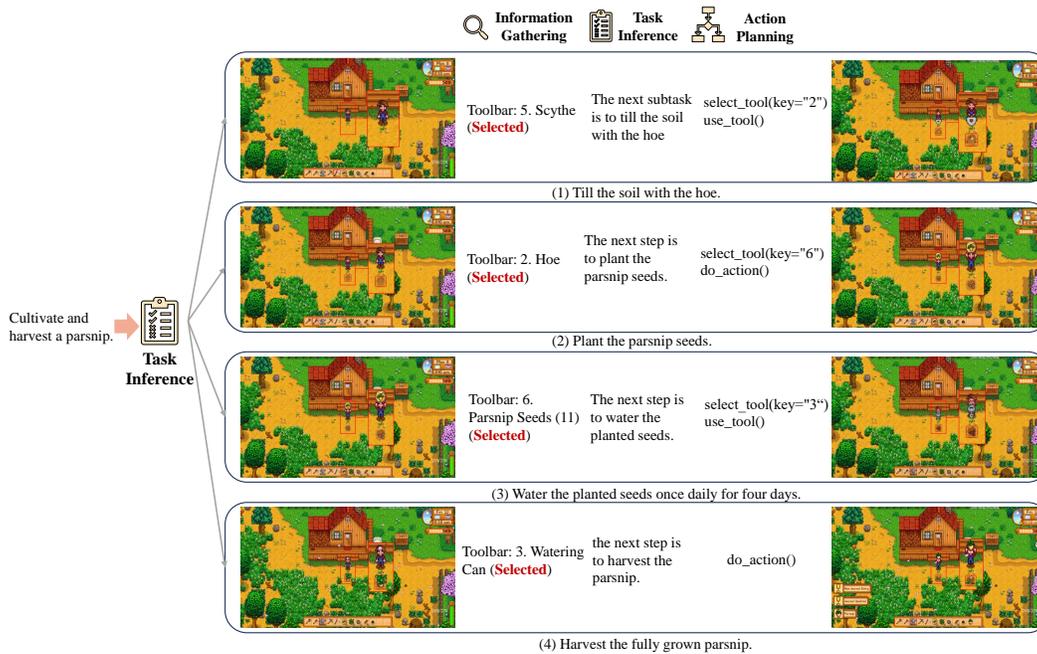


Figure 23: Case study of task inference on decomposing a task into specific sub-tasks. The complete task is to cultivate and harvest a parsnip. CRADLE decomposes the task into four sub-tasks by task inference. Only relevant modules are shown for better readability, though all modules (Figure 3) are executed per iteration.

F.5 LIMITATIONS OF GPT-4O

Fine-grained Control. Stardew Valley requires that players are positioned precisely to interact with objects, such as doors and NPCs. However, it is difficult for GPT-4o to take a pixel-level precise action. For example, GPT-4o can not take a precise movement even though the speed at which the figure moves is known. To alleviate this problem, we make some composite skills that use template-matching to complete some complex interaction tasks, such as purchasing items.

Perception in a 2D virtual world . In Stardew Valley, it's common for a character to be blocked by rocks or trees, and GPT-4o fails to tell if a character is blocked by looking at the image once, and can't predict if the next move will be blocked, which is very easy for a human to do by looking at the image. This indicates that GPT-4o is relatively weak in perceiving the virtual world in this game. In order to solve this problem, we compare the successive frames before and after in Self-Reflection to enable GPT-4o to judge the corresponding changes.

G DEALER’S LIFE 2

G.1 INTRODUCTION TO DEALER’S LIFE 2

Dealer’s Life 2 is a captivating indie simulation game developed by Abyte Entertainment. Renowned for its intricate negotiation mechanics and humorous portrayal of a pawn shop environment, the game is celebrated for its engaging gameplay that combines strategy with a quirky, cartoonish art style. As a simulation game with role-playing elements, Dealer’s Life 2 is played from a first-person perspective, utilizing a mouse for point-and-click interactions and a keyboard for price inputs. This interface facilitates item appraisals, customer interactions, and comprehensive shop management.

In the game, players assume the role of a pawn shop manager, tasked with acquiring and selling various items to make a profit while managing their store’s reputation and inventory. Players engage with a wide range of unique non-player characters (NPCs), each with their own distinct behaviors and negotiation styles. Whether bartering over the price of a rare collectible or managing unforeseen shop events, players must hone their haggling and strategic decision-making skills to succeed. Dealer’s Life 2 operates in a closed-source format with no APIs available for accessing in-game data or automating gameplay functions. This setup ensures a hands-on experience where players are immersed in the day-to-day challenges of running a pawn shop. This game environment provides a unique and entertaining setting for testifying the GCC’s haggling and strategic decision-making abilities. We run our experiments using the latest version, V. 1.013_W96 of the game.

G.2 OBJECTIVES

We concentrate on evaluating the sustained management skills required to maximize profits through buying and selling a diverse range of items from customers. Therefore, the task in this game is defined as *Weekly shop management*, *i.e.*, managing a shop for a week automatically. This game could effectively demonstrate the negotiation ability of the LMM in a trade and bargain. For example, giving an unacceptable price to the customers, *i.e.*, a pretty low price for a seller customer or a very high price for a buyer customer, could cause the deal to fail directly, which brings no profit in this situation. The key is to carefully analyze the description of the item, *e.g.*, the rarity and condition of the item, and more importantly, the response of the customer, *i.e.*, the customer’s mood changes.

Contrary to many games that feature detailed tutorials highlighting specific operations and objectives through each crucial step, Dealer’s Life 2 does not provide such guidance. This absence transforms the game into a zero-shot, hard open-world task, where the LMM must directly apply its prior knowledge of haggling and strategic decision-making to a new and unfamiliar environment. To provide readers with a clear and straightforward understanding of the task, we illustrate the typical flow of a day’s shop management through several key steps, presented in Table 10.

Table 10: Key points in the open-ended mission, *Weekly shop management* in Dealer’s Life 2. Figure 24 showcases snapshots of key points (specific sub-figures marked in parenthesis in the table).

Task: Weekly shop management	Description
Open shop (Fig. 24a)	Start a new day shop management.
Dialog (Fig. 24b)	Choose an option in a dialog.
Item Description (Fig. 24c)	View the item information
Haggle (Fig. 24d)	Give a price for the item.
Deal Result (Fig. 24e)	View the deal results.
Stats (Fig. 24f)	View shop stats.

G.3 IMPLEMENTATION DETAILS

The implementation of Dealers’ Life 2 also strictly follows the GCC framework, which includes Information Gathering, Self-Reflection, Task Inference, Skill Curation, Action Planning, and Action Execution. The details are described in Appendix D. Therefore, we emphasize the specific implementations for Dealers’ Life 2.

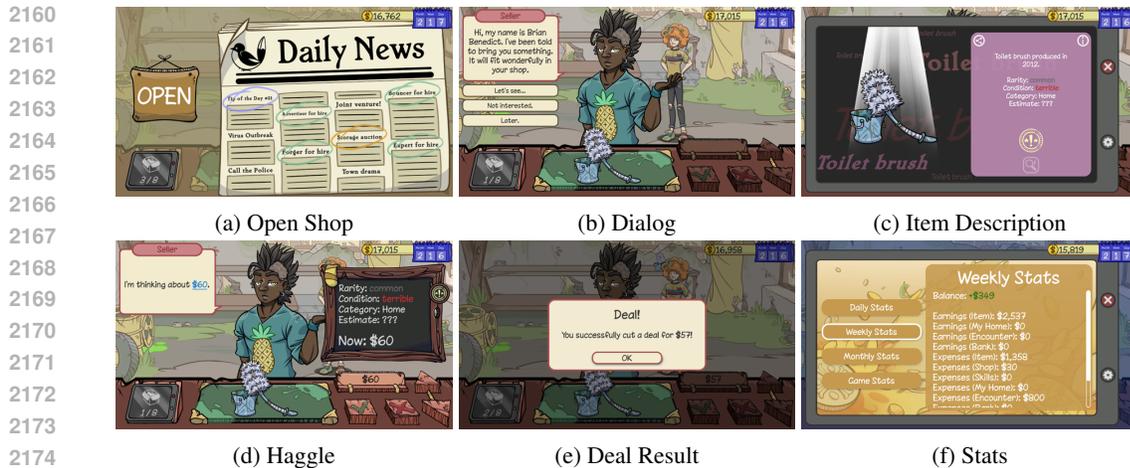


Figure 24: Image examples of key points in the open-ended task of Dealers' Life 2.

Procedural Memory. Due to the absence of a new-user guide, the LMM cannot directly and accurately know the operation method or effect of an action in the game, *e.g.*, giving the price can only use the keyboard to input an integer in an abstract box in the bottom right of the haggle screen as shown in Figure 24d, by directly observing the screen. Unless the player executes an action and observes what is happening, the player cannot know what its effect is. However, this could easily cause severe errors in an open-world environment. For example, if the player gives a price at \$100,000 for an item without knowing what the box is, it could cause the player to lose all the money. Besides, this game is very simplified with finite types of screen content and fixed buttons positions for processing the deal, where we could categorize the screen types and design general atomic skills for them. Thus, with a focus on evaluating the LMM's zero-shot haggling and strategic decision-making ability in managing a shop, we believe it is reasonable to skip the skill curation by directly setting several atomic skills as the initialization of the procedural memory, such as "process_dialog()" for clicking on the option of a dialog screen to keep the deal going on as shown in Figure 24b. The description of all the atomic skills is listed as follows:

- *open_shop()*: The function to open the dealer's shop to start dealing for today.
- *give_price(price)*: The function to give a price for the item in the deal. The price must be an integer number.
- *process_dialog()*: The function to click on to choose the first option of the dialog to make the game go on.
- *close_description_page()*: The function to close a description page showing information about the item details, daily stats, or the traits of the buyer or seller.
- *accept_deal()*: The function to click on the check mark to accept the deal on the confirmation dialog.
- *reject_deal()*: The function to click on the cross mark to reject the deal on the confirmation dialog.
- *finish_buy()*: The function to click on the ok button to finish the deal on the confirmation dialog.
- *finish_sell()*: The function to click on the ok button to finish the selling on the confirmation dialog.

Self-Reflection. Additionally, as Dealers' Life 2 has no heavy need for a long-term reflection, so we only use the first and last frame of the video as input to reduce the number of tokens used per request. Finally, this self-reflection module could help to keep the game going, instead of sticking to the same point in the game.

Action Planning. In this game, we restrict GPT-4 to output only one skill per action because it is a round-based game that does not require frequent execution of actions, and the state of the next

2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267



(a) Agent: Given that the customer is a seller and the item on offer is in bad condition with an estimated price of \$ 280, it would be wise to offer around \$150 to leave room for negotiation.



(b) Agent: Given the item's bad condition and \$280 estimate, A reasonable starting counteroffer would be 50-60% of its estimated value, and \$160 is a reasonable counteroffer.



(c) Agent: A reasonable next offer could be \$180 or \$190, aiming to reach a middle ground. I would still risk with \$160 to see if the offer is cut further.



(d) Agent: The customer is proposing her final offer at \$205. It will be necessary to accept the offer as this is still a profitable margin.

Figure 25: The reasoning of **CRADLE** in a successful deal with haggling. The price provided by **CRADLE** is keyed in by keyboard and mouse operations in the digital display box in the bottom-right corner.

time-step after an action is executed is highly uncertain, *e.g.*, the unpredictable mood changes in a customer's response.

G.4 CASE STUDIES

Here we present a few game-specific case studies to further discuss **CRADLE**'s reasoning and decision-making process in the GCC setting.

G.4.1 SUCCESSFUL NEGOTIATION

Figure 25 illustrates a successful negotiation by **CRADLE** with an NPC seller over an item valued at \$280. **CRADLE** determines a strategic starting offer by considering both the item's quality and the customer's initial proposal. Throughout subsequent negotiation rounds, **CRADLE** leverages its memory to maintain an offer close to the initially assessed \$160, applying pressure on the customer to reduce their expectations. However, **CRADLE** also demonstrates flexibility, adapting its strategy when faced with the customer's final offer—signaled by their incline to leave. This allows **CRADLE** to secure a final agreement that still yields a profitable deal.

G.4.2 UNSUCCESSFUL NEGOTIATION

Figure 26 illustrates a scenario where **CRADLE** engages in an unsuccessful negotiation. The seller consistently demands a price above the estimated value of the item, while **CRADLE**, aiming to secure a profit, steadfastly offers a price below the estimated value. A common price cannot be arrived at after rounds of negotiation. Consequently, the negotiation fails to reach an agreement, resulting in the departure of the high-expectation customer.

2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321



(a) Agent: I will offer an initial price of \$100 to reflect the poor condition while allowing room for negotiation since the condition is poor.

(b) Agent: The buyer suggests higher than the estimated price. I should offer a price lower than the item's estimated value. I will offer \$110.



(c) Agent: The seller has countered several times already, suggesting that further negotiations could result in a mutually acceptable price, possibly just a little bit below \$164.



(d) Agent: The negotiation failed. My next step is to acknowledge the failure and continue the game-play to interact with other customers to profit.

Figure 26: The reasoning of **CRADLE** in an unsuccessful deal with haggling. The price provided by **CRADLE** is keyed in by keyboard and mouse operations in the digital display box in the bottom-right corner.

G.4.3 ACQUIRING AND SELLING OF A COUNTERFEIT ITEM

Figure 27 depicts a complete buying and selling cycle involving an item later identified as counterfeit by a buyer. Notably, **CRADLE** initially acquires the item without a known price estimate, basing its purchase solely on the seller's initial offer. In subsequent transactions, although **CRADLE** recognizes the item's potential forgery, it continues to trade the item in pursuit of maximizing profit. This strategy yields a substantial profit margin of 145%. However, it also poses a risk to the in-game character's reputation, highlighting a strategic decision made under the directive to prioritize profit.

G.5 QUANTITATIVE EVALUATION

The effectiveness of **CRADLE** in terms of game performance is evaluated through experiments conducted over a 7-day gaming scenario using nine quantitative financial metrics. A comprehensive description of these metrics is provided in Section G.6.

G.6 EVALUATION METRICS

Assuming the buying price for item i is denoted by B_i , the selling price by S_i , the market valuation by V_i , and the number of successfully traded items is n . To evaluate **CRADLE**'s profitability and performance in Dealer's Life 2, we use the following evaluation metrics:

- **Turnover Rate (TR)** can be calculated as the ratio of the number of successfully traded items to the total number of items considered (both successfully and unsuccessfully traded). It reflects the Agent's ability to successfully complete transactions and can indicate operational efficiency, market competitiveness, and negotiation effectiveness. The calculation formula is $TR = \frac{n}{n+m}$.
- **Gross Profit Margin (GPM)** is the ratio of gross profit to sales revenue, reflecting the dealer's direct profit capability after selling items. The calculation formula is $GPM = \frac{\sum_{i=1}^n S_i - B_i}{\sum_{i=1}^n S_i}$.

2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375



(a) Agent: Given that we do not know the estimated price, I proceed with 70%-80% of the asked price, around \$4,200 to \$4,480.



(b) Agent: Since the seller is willing to lower the price, the current counteroffer may still be negotiated.



(c) Agent: The counteroffer of \$4,900 seems reasonable, but I can go further for the best possible price.



(d) Agent: The response will be final. Given this, accepting the deal is a sensible to secure the item.



(e) Agent: As I am selling the item, I'll try for a higher price for higher profits.



(f) Agent: I note that the customer rejected my counter-offer, calling the item fake.



(g) Agent: The seller is in an urgent need to buy the jeweled necklace.



(h) Agent: From the warning window, I am told that this necklace is fake.



(i) Agent: Though the jeweled necklace is fake, accepting this offer provides substantial profit with the buyer's offer at \$11,400.



(j) Agent: Despite the successful sale at a good price, the item is revealed as a fake. The added profit is good (+145%)

Figure 27: Case in acquiring and selling an item for multiple attempts with reasoning, and dealing with unexpected information on the authenticity. The price provided by CRADLE is keyed in by keyboard and mouse operations in the digital display box in the bottom-right corner.

- 2376 • **Return on Investment (ROI)** is the ratio of profit to investment, used to measure the dealer’s
 2377 return on investment for items. The calculation formula is $ROI = \frac{\sum_{i=1}^n S_i - B_i}{\sum_{i=1}^n B_i}$.
 2378
- 2379 • **Valuation Deviation (VD)** reflects the difference between the selling price and the market
 2380 valuation, used to evaluate the reasonableness of the pricing strategy. It is denoted as $VD =$
 2381 $\frac{\sum_{i=1}^n S_i - V_i}{\sum_{i=1}^n V_i}$.
 2382
- 2383 • **Buying Price to Valuation Ratio (BPVR)** can help determine whether the buying price is lower
 2384 than the market valuation, reflecting the success of the procurement. The calculation formula is
 2385 $BPVR = \frac{\sum_{i=1}^n B_i}{\sum_{i=1}^n V_i}$.
- 2386 • **Selling Price to Valuation Ratio (SPVR)** reflects the selling price relative to the market
 2387 valuation, helping to assess the success of the sales. The calculation formula is $SPVR = \frac{\sum_{i=1}^n S_i}{\sum_{i=1}^n V_i}$.
 2388
- 2389 • **Average Profit Rate (APR)** reflects the overall profitability of the dealer on items. Assuming
 2390 the return rate for item i is $\frac{S_i - B_i}{B_i}$, the calculation formula of average return rate is denoted as
 2391 $APR = \frac{1}{n} \sum_{i=1}^n \frac{S_i - B_i}{B_i}$.
- 2392 • **Maximum Return Rate (MRR)** is the highest return rate among all items. The calculation
 2393 formula is $MRR = \max(\frac{S_1 - B_1}{B_1}, \frac{S_2 - B_2}{B_2}, \dots, \frac{S_n - B_n}{B_n})$.
 2394
- 2395 • **Minimum Return Rate (mRR)** is the lowest return rate among all items. The calculation
 2396 formula is $mRR = \min(\frac{S_1 - B_1}{B_1}, \frac{S_2 - B_2}{B_2}, \dots, \frac{S_n - B_n}{B_n})$.
 2397

2398 Table 11: Performance of CRADLE with GPT-4o in Dealer’s Life 2 gameplay. “# attempts” represents the total
 2399 number of all negotiation attempts on items, including both successful and unsuccessful transactions.

Exp	# attempts	TR↑	GPM↑	ROI↑	VD↑	BPVR↓	SPVR↑	APR↑	MRR↑	mRR↑
2401 01	13	92.86	20.38	25.60	13.17	90.10	113.17	42.97	105.56	0.00
2402 02	12	91.67	18.89	23.30	23.30	100.00	123.30	17.98	97.76	0.00
2403 03	12	83.33	26.81	36.63	34.39	98.36	134.39	38.68	127.27	-8.06
2404 04	9	100.00	49.35	87.45	80.69	93.53	165.74	66.45	145.16	0.00
2405 05	12	100.00	20.61	25.25	25.25	100.00	125.25	23.08	44.33	0.00
2406 Avg.	11.6	93.57	27.21	39.65	35.36	96.40	132.37	37.83	104.02	-1.61

2408 H CITIES: SKYLINES

2409 H.1 INTRODUCTION TO CITIES: SKYLINES

2410
 2411
 2412 Cities: Skylines is a single-player open-ended city-building simulation game developed by Colossal
 2413 Order. In the game, players assume the role of a city planner, tasked with building and managing
 2414 various aspects of a city to ensure its growth and prosperity. Players engage with a wide range of
 2415 urban challenges, from managing traffic flow to balancing the budget, and from providing essential
 2416 services to fostering a vibrant economy. Each decision impacts the city’s development, requiring
 2417 players to hone their planning and strategic decision-making skills to succeed. Effective city man-
 2418 agement leads to thriving neighborhoods, a growing economy, and high citizen satisfaction, while
 2419 mismanagement can result in traffic congestion, service shortages, and a decline in population and
 2420 reputation. Proper planning and responsive governance are crucial for a city that flourishes and
 2421 remains appealing to its residents and visitors.

2422 As the city’s infrastructure and various supporting resources are well-developed, it can attract more
 2423 people. And a larger population brings more tax revenue and also brings greater expenses to the
 2424 city’s operations. If operated properly, the increasing population can continuously unlock richer ur-
 2425 ban facilities; if operated improperly, such as road congestion, insufficient services, housing short-
 2426 age, water and electricity shortage, noise pollution, water pollution, excessive garbage, disease, fire
 2427 Situation, etc., will all lead to population decline.

2428 This game could be used to evaluate agents’ strategies in managing urban development and resource
 2429 allocation. By simulating different scenarios, agents can experiment with various policies and infras-
 structural changes to see their impacts on the city’s growth and sustainability. Effective strategies may

2430 involve optimizing public transportation systems to reduce road congestion, investing in renewable
 2431 energy sources to prevent power shortages, and implementing comprehensive waste management
 2432 programs to handle excessive garbage. It offers a risk-free environment to test innovative ideas
 2433 and learn from the consequences of their actions, ultimately promoting a deeper understanding of
 2434 sustainable urban development.

2435 Though this game is ranked very positive on Steam, it is notorious for its extremely high difficulty
 2436 for beginners, as it lacks a detailed tutorial in the beginning, which introduces more challenges for
 2437 **CRADLE** to deal with. On the other side, Although the successor, *Cities: Skylines 2*, simplified the
 2438 controls and provided a detailed tutorial for beginners, it became notorious for poor optimization
 2439 and frequent crashes that caused computer blue screens. As a result, we had to back to using *Cities:*
 2440 *Skylines 1* instead of 2. And we do not apply any modes to the game. We use the latest version of
 2441 the game (version 1.17.1-f4).

2442

2443 H.2 OBJECTIVES

2444

2445 Our mission is to build cities so that they can support as many people as possible. Maps in this game
 2446 are usually very large, which usually costs human players dozens of hours to cover all areas. Besides,
 2447 the technology tree unlocks as the population grows, which requires multiple turns of planning and
 2448 building. In this work, we simplified the problem by starting the game near the water and fixing the
 2449 viewpoint (as shown in Figure 28), so that **CRADLE** can leverage the pixel position in the screenshot
 2450 to locate the position of placed buildings and facilities. Agents start with a plot of land, which is
 2451 equipped with an entry and an exit from a major highway, providing crucial access for future traffic
 2452 flow, and proximity to the water source, which is essential for the city’s water supply needs. And we
 2453 focus on the first turn of planning, i.e., pause the game and stop the passage of the in-game time, use
 2454 the initial starting funds of €70,000 and the most basic road, water, and electricity facilities provided
 2455 at the beginning of the game, which is enough to achieve the first milestone, *Little Hamlet* with
 2456 the population of 440 in the game. Then what kind of city can **CRADLE** create? Can this city ensure
 2457 water and electricity supply to keep functioning normally while reasonably dividing residential,
 2458 commercial, and industrial zones? A run is terminated when it reaches the maximal steps, 1000, or
 2459 the budget is used up (less than € 1000).

2459



2460

2469 Figure 28: Demonstration for the initialization location of our mission in *Cities: Skylines*, which is near the
 2470 river and contains the entry and exit of the highways.

2471

2472

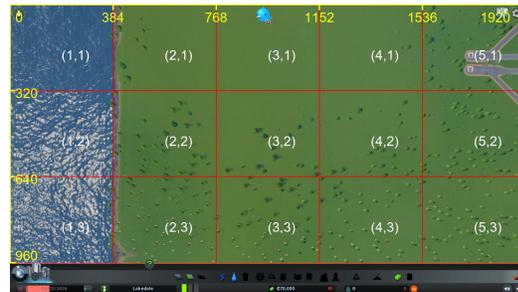
2473 H.3 EVALUATION METRIC

2474

2475 To measure the completeness of the city built by the agent, we design the following preliminary
 2476 metrics:

2477

- 2478 • **Roads in closed loop:** Whether the road is a closed loop, which is crucial for ensuring
 2479 smooth traffic flow and is beneficial for the city’s future development.
- 2480 • **Sufficient water supply:** To ensure a sufficient water supply, the player needs to construct
 2481 a water pumping station at the shoreline and then use water pipes to cover every district
 2482 along the roads. To manage the effluent effectively, the other end of the water pipe network
 2483 must be equipped with the water drain pipe which is also required to be placed near the
 shoreline.



2460

2469 Figure 29: Visual prompting methods used in *Cities: Skylines*. The full screenshot is divided into 3×5 grids
 2470 and each grid is assigned a unique white coordinate.

2471

2472

2473 H.3 EVALUATION METRIC

2474

2475 To measure the completeness of the city built by the agent, we design the following preliminary
 2476 metrics:

2477

- 2478 • **Roads in closed loop:** Whether the road is a closed loop, which is crucial for ensuring
 2479 smooth traffic flow and is beneficial for the city’s future development.
- 2480 • **Sufficient water supply:** To ensure a sufficient water supply, the player needs to construct
 2481 a water pumping station at the shoreline and then use water pipes to cover every district
 2482 along the roads. To manage the effluent effectively, the other end of the water pipe network
 2483 must be equipped with the water drain pipe which is also required to be placed near the
 shoreline.

- 2484 • **Sufficient electricity supply:** Both zones and water facilities need electricity to power.
2485 To provide sufficient electricity supply, the player can build a coal power plant or wind
2486 turbine. Considering coal power plants cost too much and will create heavy pollution,
2487 wind turbines combined with the power lines are a better choice at the beginning. The
2488 electricity area extends automatically based on the presence of buildings and infrastructure
2489 that consume electricity.
- 2490 • **Zones Coverage > 90%:** The built two-lane road will provide empty space for the devel-
2491 opment of zones, *i.e.*, residential zone, commercial zone and industrial zone. Residential
2492 zones provide houses for people to live in, which is the most essential zone to increase
2493 the population. Commercial zones provide places for small businesses, shops, and services
2494 produced in the industrial zones or imported. Industrial zones provide jobs for the residents
2495 and products for commercial buildings, which is also important to attract more people to
2496 move to the city. This metric is used to evaluate whether 90% of the available areas are
2497 covered by the zones. The agent needs to reasonably allocate the areas and proportions of
2498 various zones to achieve better city development and attract a larger population.
- 2499 • **Maximal population:** After **CRADLE** finishes building, we will unpause the game and
2500 start the simulation. Then houses start to be built and residents start to move in. We will
2501 record the maximal population during the simulation as the value for this metric.
- 2502 • **Maximal population with assistance:** We find that cities built by **CRADLE** manage to
2503 meet most of the requirements but suffer a significant population loss due to a few easy-to-
2504 fix mistakes. So after **CRADLE** finishes the design of the city, we apply human assistance
2505 that attempts to address these small mistakes within 3 unit operations (building or removing
2506 a road/facility/a place of zones is counted as one unit operation). We will also record the
2507 maximum population during the simulation in the city with human assistance.

2508 H.4 IMPLEMENTATION DETAILS

2509 The implementation of Cities: Skylines also strictly follows the GCC framework, which includes
2510 Information Gathering, Self-Reflection, Task Inference, Skill Curation, Action Planning and Action
2511 Execution. The details are described in Appendix D. Therefore, we emphasize the specific design
2512 for Cities: Skylines.

2514 **Pause.** Since the game is stopped before starting the simulation, there is no need to unpause and
2515 pause the game while executing actions.

2516 **Visual Prompting.** As shown in Figure 29, similar to Stardew Valley, we divide each screenshot
2517 into 3×5 grids with an axis based on the resolution of the game screen. Then **CRADLE** can utilize
2518 the pixel-level position in the screenshot to locate the building and facility. We empirically find that
2519 this visual prompting method can result in a more precise control of GPT-4o.

2520 **Information Gathering.** In Cities: Skylines, the game’s perspective is typically adjustable, al-
2521 lowing players to zoom in and out, rotate, and pan across their cityscape to get a detailed view of
2522 their urban development. To ensure consistency and ease of navigation for GPT-4o, we have locked
2523 the camera angle and applied a visual prompting method to enhance GPT-4o’s visual understand-
2524 ing. Besides, we use GPT-4o to extract key information, such as budget, population, construction
2525 information and error messages, in the game.

2526 It is worth noting that in this module, we feed the original screenshot to GPT-4o, rather than the
2527 augmented screenshot with axis and coordinates. We find that the numbers and lines may cover some
2528 key information and result in wrong OCR recognition. For example, the construction information,
2529 "Estimated Production: 120,000m³/week" may be mistakenly interpreted as "Estimated Production:
2530 000,000m³/week" by GPT-4o, due to interference from the lines and numbers. This construction
2531 information is a key signal for the suitable place of the water pumping station. For the other modules,
2532 we feed GPT-4o with the augmented screenshots.

2533 **Self-Reflection.** Since actions in this game are very short, and each of them has a significant effect
2534 shown in the last screenshot. We only use the first screenshot and the last screenshot of the video clip
2535 as input to this module, which is proved to be enough for not missing any important information.

2536 **Task Inference.** Due to the lack of a detailed tutorial, we have to provide a draft blueprint for
2537 the GPT-4o as the plan at the beginning to help GPT-4o to determine the next step to do. This

plan provides guidance to the orders of building each facility and how to build a closed road, how to ensure water and electricity supply and zone placement. Even so, we find that GPT-4o failed frequently to follow the plan, resulting in the lack of building some important facilities, like water pumping stations.

Skill Curation. Due to the lack of detailed tutorials in the game, we generate the skills through self-exploration in this game. The skill generation basically involves manipulating the toolbar to understand the items on it. The pseudo-code for skill generation is described in Algorithm 1. This process leverages SAM for objective grounding and GPT-4o to gather information about the objects provided by the game, subsequently generating skills based on a predefined template. An example of the process is shown in Fig 30, 31, 32, 33, 34 and 35.



Figure 30: The toolbar in Cities: Skylines

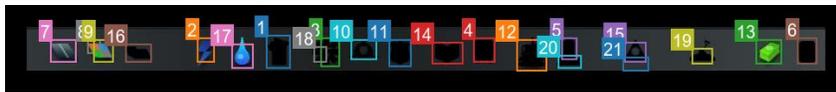


Figure 31: The grounding result of the toolbar in Cities: Skylines



Figure 32: When hovering the mouse over a toolbar item, the pop-up description is "Water & Sewage". The skill generated is then called "open_water_sewage_menu".

Figure 33: When hovering the mouse over a toolbar item, the pop-up description is "Education - Reach a population of 440". As this is not selectable for now, GPT-4o does not generate a new skill for it.

Action Planning. In this game, we only let GPT-4o output one skill for each action since we observe that GPT-4o tends to output *try_place* and *confirm_placement* together if we allow it to output and execute multiple skills in one action, which is against the intention of our design for the *try_place* action.

Procedure Memory. Skills generated through self-exploration are listed below:

- *open_roads_menu()*: The function to open the roads options in the lower menu bar for further determination of which types of roads to build.
- *open_electricity_menu()*: The function to open the electricity options in the lower menu bar for further determination of which types of power facility to build.
- *open_water_sewage_menu()*: The function to open the water and sewage options in the lower menu bar for further determination of which types of water and sewage to build.
- *open_zoning_menu()*: The function to open the zoning options in the lower menu bar for further determination of which types of zonings to build.
- *try_place_two_lane_road*(x_1, y_1, x_2, y_2): Previews the placement of a road between two specified points, (x_1, y_1) and (x_2, y_2) , with x_1, y_1 being the coordinate of start point of the road, and (x_2, y_2) being the coordinate of end point of the road. This function does not actually construct the road, but rather displays a visual representation of where the road would be placed if confirmed.

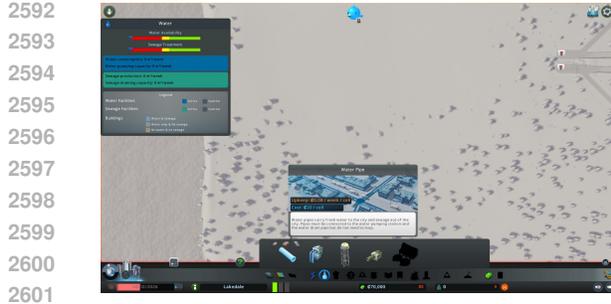


Figure 34: The Water & Sewage menu is opened by executing the new skill "open_water_sewage_menu". The Agent then hovers the mouse over a second-level toolbar item, the pop-up description is "Water Pipe", and the generated skill is called "try_place_water_pipe".

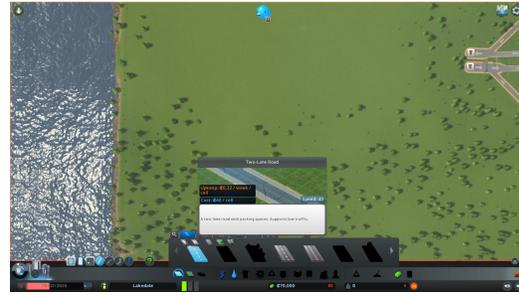


Figure 35: The Roads menu is opened by executing the new skill "open_roads_menu". The Agent then hovers the mouse over a second-level toolbar item, the pop-up description is "Two-Lane Road", and the generated skill is called "try_place_two_lane_road".

Algorithm 1: Skill Generation

Input: Toolbar with objects, Skill template

Output: Procedure memory with generated skills

```

1 Initialize procedure memory;
2 for each object in the toolbar do
3   Hover the mouse on the object to get the description;
4   Generate skill using GPT-4o based on the object description and the skill template;
5   Store generated skill in procedure memory;
6   Execute the generated skill to enter the second-level toolbar;
7   for each object in the second-level toolbar do
8     Hover the mouse on the object to get the description;
9     Generate skill using GPT-4o based on the object description and skill template;
10    Store generated skill in procedure memory;
11 return procedure memory

```

- $try_place_wind_turbine(x, y)$: Previews the placement of a wind turbine on point, (x, y) . This function does not actually construct the wind turbine, but rather displays a visual representation of where the wind turbine would be placed if confirmed.
- $try_place_water_pumping_station(x, y)$: Previews the placement of a water pumping station on point, (x, y) . This function does not actually construct the water pumping station, but rather displays a visual representation of where the water pumping station would be placed if confirmed.
- $try_place_water_pipe(x_1, y_1, x_2, y_2)$: Previews the placement of a water pipe between two specified points, (x_1, y_1) and (x_2, y_2) . This function does not actually construct the water pipe, but rather displays a visual representation of where the water pipe would be placed if confirmed.
- $try_place_water_drain_pipe(x, y)$: Previews the placement of a water drain pipe on point, (x, y) . This function does not actually construct the water drain pipe, but rather displays a visual representation of where the water drain pipe would be placed if confirmed.
- $try_place_commercial_zone(x_1, y_1, x_2, y_2)$: Previews the placement of a commercial zone within a rectangular region with diagonal corners at (x_1, y_1) and (x_2, y_2) . This function does not actually construct the commercial zone, but rather displays a visual representation of where the commercial zone would be placed if confirmed.
- $try_place_industrial_zone(x_1, y_1, x_2, y_2)$: Previews the placement of a industrial zone within a rectangular region with diagonal corners at (x_1, y_1) and (x_2, y_2) . This function does not actually construct the industrial zone, but rather displays a visual representation of where the industrial zone would be placed if confirmed.

- $try_de_zone(x_1, y_1, x_2, y_2)$: The function to remove the zone in the game. The zone must cover the road.
- $confirm_placement()$: The function to confirm the placement and build the object after the try_place_object function.
- $cancel_placement()$: The function to cancel the placement of the object after the try_place_object function.

Episodic Memory. Besides the common information to store in the episodic memory. We initialize the memory with the coordinates of the entry and exit of the highway. Then **CRADLE** is able to extend the roads according to these two points at the beginning. When a road or a facility such as wind turbine, water pumping station, water drain pipe and water pipe is placed on the map, the corresponding coordinates will also be stored in the memory for future development of the city.

H.5 CASE STUDIES

H.5.1 FAILURE FOR ROAD BUILDING.

As shown in Figure 36, sometimes GPT-4o will build a long road, which ends on the top of water. The recorded endpoint of the road is actually the projection of the road on the sea level, resulting in the offset from the projection point and the real endpoint of the road. It leads to the failure of extending the road to the other places.

Figure 36b, 36c, 36d and 36e tells a story that GPT-4o sometimes forgets to confirm the placement (from 36c to 36d) and directly moves to the next step of building the next road (from 36d to 36e), resulting in the disconnection of the roads.

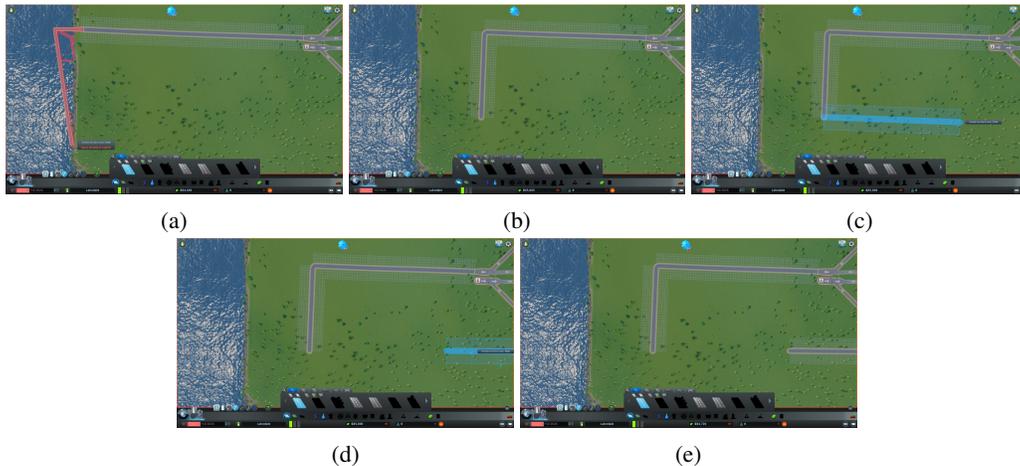
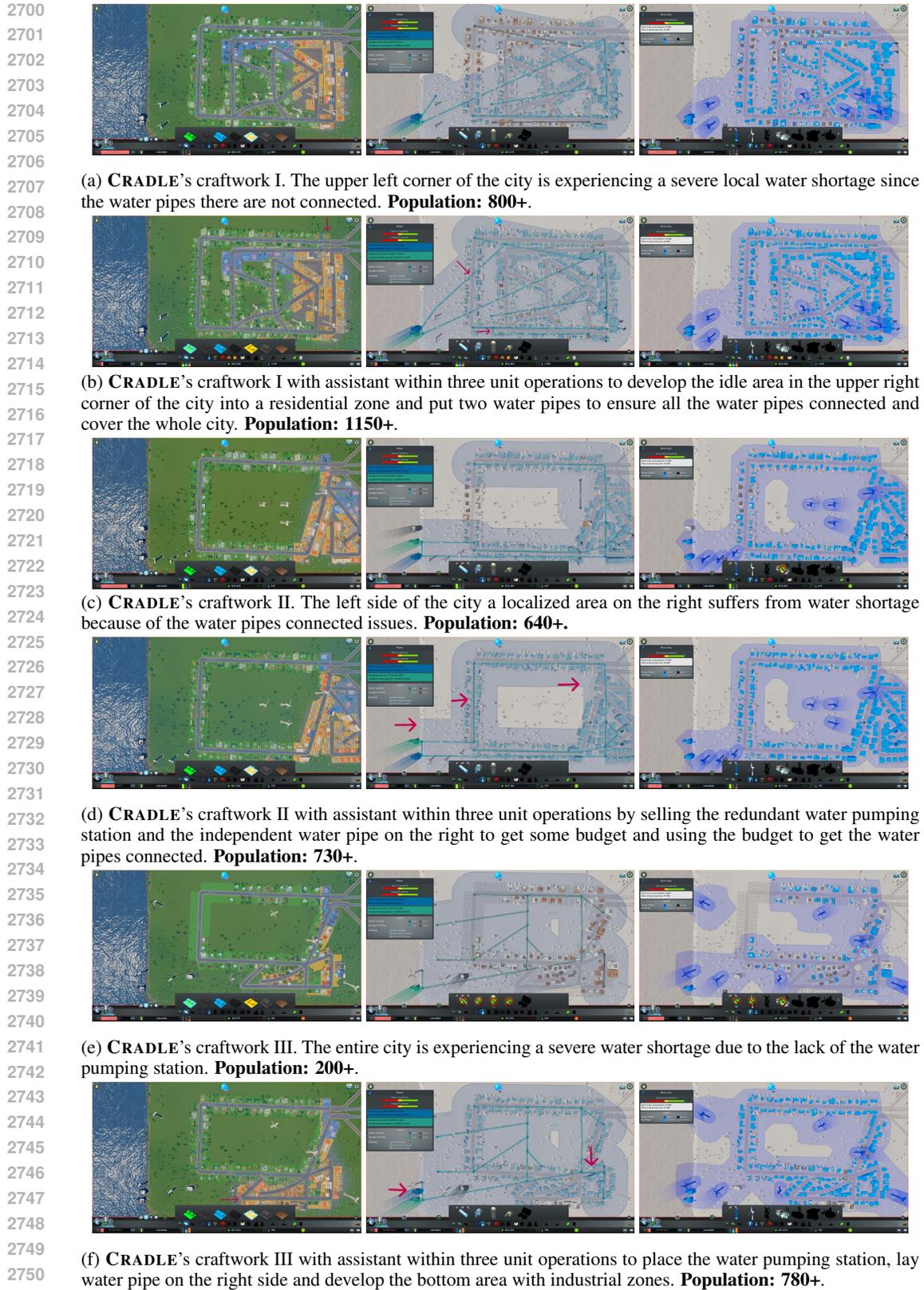


Figure 36: Failure cases of building roads in a closed loop. Figure 36a shows that the road is built over the water and is difficult to continue. Figure 36b, 36c, 36d and 36e tells a story that GPT-4o sometimes forgets to confirm the placement (from 36c to 36d) and directly moves to the next step of building (from 36d to 36e), resulting in the disconnection of the roads.

H.5.2 FAILURE FOR SUFFICIENT WATER SUPPLY.

Figure 37 displays three cases where **CRADLE** fails to ensure the water supply due to the disconnection of water pipes and the missing water pumping station. All of them can be fixed within three unit operations. As shown in Figure 37b and 37f, we observe a significant increase in the population if these mistakes are fixed, which proves that **CRADLE** already has the ability to build a reasonable city but some minor adjustments are needed.



2752 Figure 37: Demonstrations of three cities built by **CRADLE** in zoning view (left), water view (middle) and electricity view (right). Figures 37b, 37d, 37f show the cities with human assistance to address construction issues (shown in red arrow). Populations shown in the figures are close to but not exactly the maximal population since they are changed dynamically.

2753

2754 I SOFTWARE APPLICATIONS

2755

2756 I.1 SELECTED SOFTWARE APPLICATIONS

2757

2758 Besides targeting complex digital games, **CRADLE** also includes an initial benchmark task set across
 2759 diverse software applications. The selected applications include Chrome, Outlook, Feishu, CapCut,
 2760 and Meitu. These applications cover popular applications for daily tasks in different usage cate-
 2761 gories, such as web browsing, communication, work, and media manipulation. Table 12 shows the
 2762 exact application versions benchmarked in this paper. Five distinct tasks were designed for each ap-
 2763 plication to represent their target domains and explore the difficulties posed to LMM-based agents
 2764 and analyze their limitations. Figure 9 shows an overview of all tasks across applications and Ta-
 2765 bles 13 and 14 detail each task.

2766 Chrome and Outlook were selected as common representatives for web browsing and e-mail, with
 2767 well-known functionality and UI design. CapCut and Meitu are two popular media editing applica-
 2768 tions for video/image editing with their own interaction styles. Lastly, Feishu (also known as Lark)
 2769 is an office collaboration and productivity application, which includes messaging, calendar/meet-
 2770 ings, and approval workflows. It represents a complex business application that doesn't strictly
 2771 follow OS-specific UI guidelines. To the best of our knowledge, this is the **first agent** targeting
 2772 applications like CapCut, Meitu, and Feishu.

2773 I.1.1 BRIEF DESCRIPTIONS

2774

2775 **Chrome** is a web browser developed by Google. It allows
 2776 users to access and utilize online resources through activi-
 2777 ties such as browsing websites, streaming videos, and us-
 2778 ing web applications. Additionally, users can customize
 2779 their browsing experience with various extensions, man-
 2780 age bookmarks and passwords, and synchronize their data
 2781 across multiple devices for seamless access.

2782 **Outlook** is an application by that allows users to manage
 2783 emails, calendars, contacts, and tasks. It includes tools
 2784 for communication and scheduling through features such
 2785 as sending and receiving emails, setting up meetings, and
 2786 keeping track of appointments. Additionally, users can customize their experience and integrate
 2787 Outlook with other Microsoft Office applications.

2788 **CapCut** is a popular video editing application developed by ByteDance. It provides easy-to-use
 2789 editing tools and enables users to create quality videos with a range of advanced features. Cap-
 2790 Cut offers a set of editing tools, including trimming, cutting, merging, and splitting video clips; the
 2791 application of various effects, filters, and transitions; as well as adjusting speed, and adding music
 2792 or text overlays.

2793 **Meitu** is a photo editing application. It is designed to cater to a broad audience and enables users
 2794 to enhance and transform their photos with minimal effort. Meitu offers editing tools, including
 2795 basic adjustments like cropping, rotating, and resizing, as well as advanced features such as beauty
 2796 retouching, filters, and special effects. Additionally, Meitu offers a wide range of stickers, frames,
 2797 and text options to further personalize photos.

2798 **Feishu**, also known as Lark, is a business communication and collaboration platform by ByteDance.
 2799 It integrates various tools for office workflows and project management. Feishu offers a wide array
 2800 of functionalities, including instant messaging, video conferencing, file sharing, and collaboration
 2801 within the app. It also includes an integrated calendar, which helps users schedule and manage
 2802 meetings and events, and task management tools that allow users to assign and track tasks.

2803

2804 I.2 SOFTWARE TASKS

2805

2806 For each of the five applications, we selected a set of representative tasks for their respective do-
 2807 mains. For example, search, navigation, and settings tasks on Chrome; sending, searching, and
 deleting emails, plus changing settings on Outlook; basic video and image editing operations on

Table 12: Exact software versions utilized in the described experiments. Similar versions should behave similarly.

Software	Version
Chrome	125.0.6422.142
Outlook	1.2024.529.200
CapCut	4.0.0
Meitu	7.5.6.1
Feishu	7.19.5

2808 Table 13: Task Descriptions for Chrome, Outlook, and CapCut. *Difficulty* refers to how hard it is for our agent
 2809 to accomplish the corresponding tasks. Figures 38, 39, and 40 illustrate each task (specific sub-figures marked
 2810 in parenthesis in the left-most column along with task name).

2811	2812	2813	2814
Software	Description	Difficulty	
Chrome			
2814	2815	2816	2817
Download Paper (Fig. 38a)	Search for an article with a title like $\{paper_title\}$ and download its PDF file.	Hard	
2816	2817	2818	2819
Post in Twitter (Fig. 38b)	Post "It's a good day." on my Twitter.	Hard	
2817	2818	2819	2820
Open Closed Page (Fig. 38c)	Open the last closed page.	Easy	
2818	2819	2820	2821
Go to Profile (Fig. 38d)	Find and navigate to $\{person_name\}$'s homepage on GitHub.	Medium	
2819	2820	2821	2822
Change Mode (Fig. 38e)	Customize Chrome to dark mode.	Medium	
Outlook			
2822	2823	2824	2825
Send New E-mail (Fig. 39a)	Create a new e-mail to $\{email_address\}$ with subject "Hello friend" and send it.	Medium	
2824	2825	2826	2827
Empty Junk Folder (Fig. 39b)	Open the junk folder and delete all messages in it, if any.	Medium	
2825	2826	2827	2828
Reply to Person (Fig. 39c)	Open an e-mail from $\{person_name\}$ in the inbox, reply to it with "Got it. Thanks.", and click send.	Medium	
2826	2827	2828	2829
Find Target E-mail (Fig. 39d)	Find the e-mail whose subject is "Urgent meeting" and open it.	Easy	
2827	2828	2829	2830
Setup Forwarding (Fig. 39e)	Set up email forwarding for every email received to go to $\{email_address\}$.	Medium	
2828	2829	2830	2831
CapCut			
2831	2832	2833	2834
Create Media Project (Fig. 40a)	Create a new project, then import $\{video_file_name\}$ to the media, click the "Audio" button to add music to the timeline, and finally export the video.	Hard	
2832	2833	2834	2835
Add Transition (Fig. 40b)	Open the first existing project. Switch to Transitions panel. Drag a transition effect between the two videos, and then export the video.	Medium	
2834	2835	2836	2837
Crop by Timestamp (Fig. 40c)	Delete the video frames after five seconds and then before one second in this video, and then export the video.	Medium	
2835	2836	2837	2838
Add Sticker (Fig. 40d)	Open the first existing project. Switch to Stickers panel. Drag a sticker of a person's face to the video, and then export the video.	Hard	
2836	2837	2838	2839
Crop by Content (Fig. 40e)	Crop the video when the ball enters the goal, and then export the video.	Very hard	
2837	2838	2839	2840

2849
 2850
 2851
 2852
 2853
 2854
 2855 CapCut and Meitu (*e.g.*, adding special effects and creating a collage); and communication and organization operations on Feishu. Tables 13 and 14 describe in detail the 25 tasks CRADLE performs and analyzes on the five selected applications; also illustrated in Figures 38, 39, 40, 41, 42, and 9.

2858 It is worth noting that we add a *special* task on CapCut to demonstrate the agent's ability for tool use. In this task, a pre-defined skill uses GPT-4o as a tool for video understanding capabilities. The skill can be selected to answer content-based questions about a video (*e.g.*, "when the ball enters the goal") and the response be used during task completion. This task is illustrated in detail in Figure 49.

2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915

Table 14: Task Descriptions for: Meitu, and Feishu. *Difficulty* refers to how hard it is for our agent to accomplish the corresponding tasks. Figures 41, and 42 illustrate each task (specific sub-figures marked in parenthesis in the left-most column along with task name).

Software	Description	Difficulty
Meitu		
Apply Filter (Fig. 41a)	Apply a filter from Meitu to $\{picture_file_name\}$ and save the project.	Easy
Cutout (Fig. 41b)	Cutout a person from $\{picture_file_name\}$ and save the project.	Easy
Add Sticker (Fig. 41c)	Add a flower sticker to $\{picture_file_name\}$ and save the picture.	Middle
Create Collage (Fig. 41d)	Make a collage using 3 pictures and save the project.	Hard
Add Frame (Fig. 41e)	Add a circle-shaped frame to $\{picture_file_name\}$ and save the picture.	Hard
Feishu		
Create Appointment (Fig. 42a)	Create a new appointment in my calendar any-time later today with title "Focus time".	Hard
Message Contact (Fig. 42b)	Please send a "Hi" chat message to $\{contact_name\}$.	Easy
Send File (Fig. 42c)	Send the AWS bill file at $\{pdf_path\}$ in a chat with $\{contact_name\}$.	Hard
Set User Status (Fig. 42d)	Open the user profile menu and set my status to "In meeting".	Medium
Start Video Conference (Fig. 42e)	Create a new meeting and meet now.	Easy

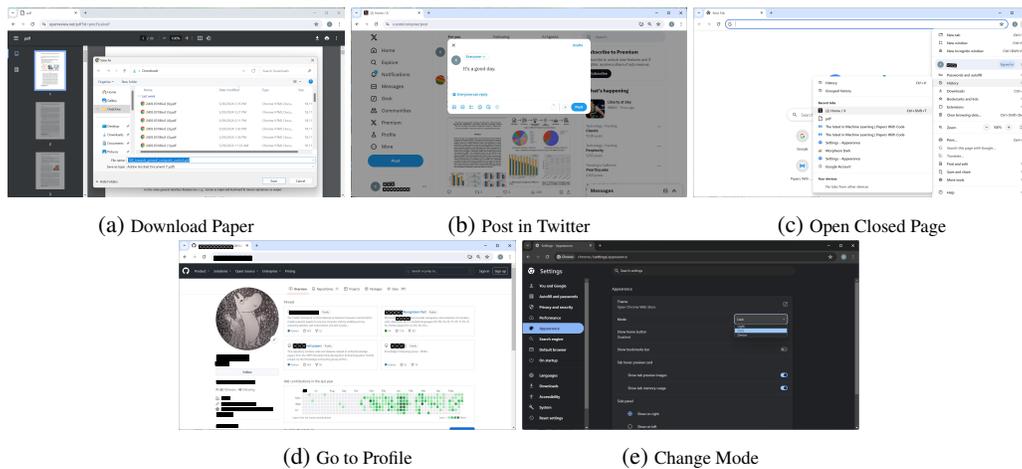


Figure 38: Screenshots of Chrome tasks.

2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969

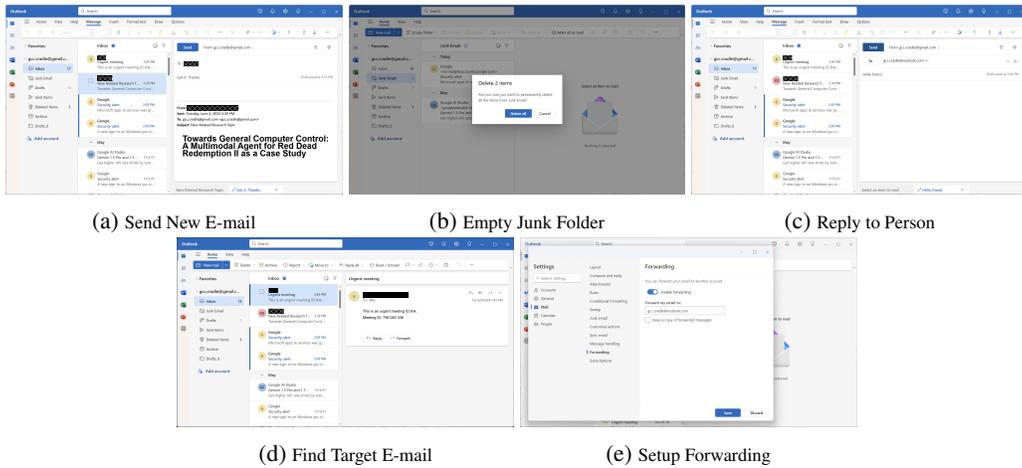


Figure 39: Screenshots of Outlook tasks.



Figure 40: Screenshots of CapCut tasks.

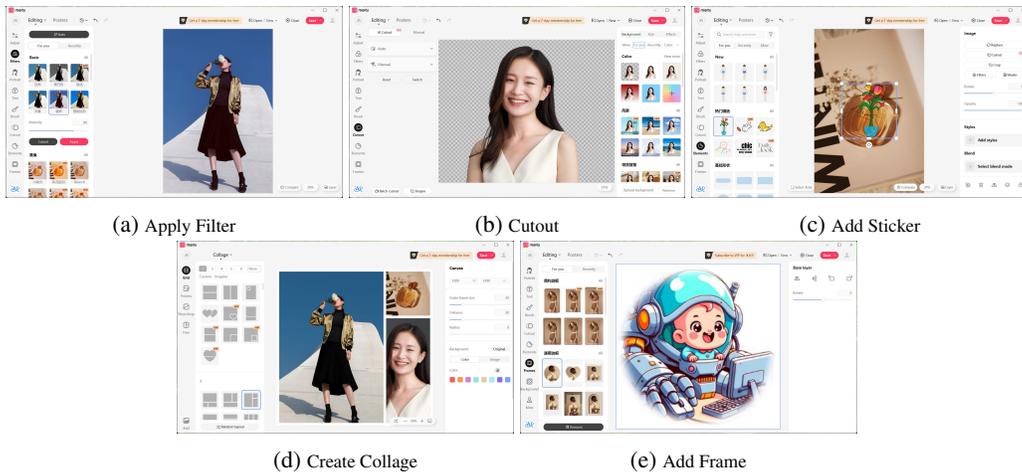


Figure 41: Screenshots of Meitu tasks.

2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023

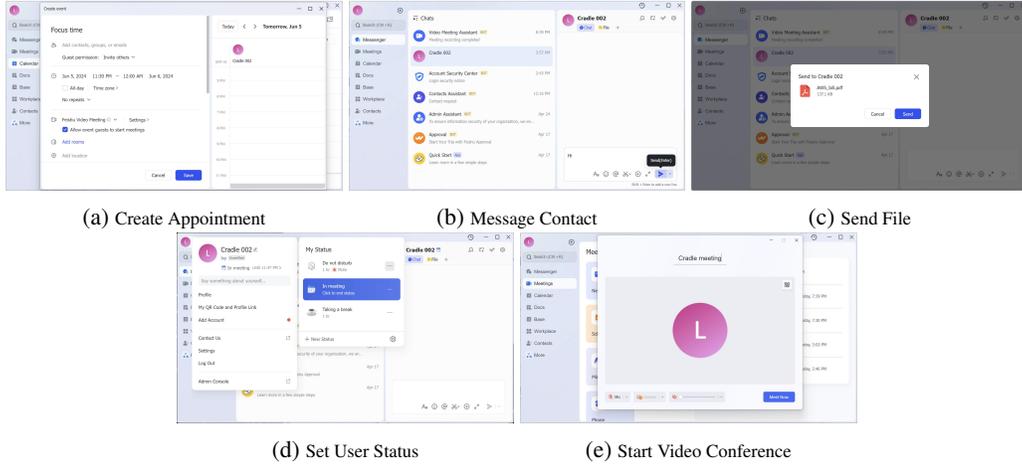


Figure 42: Screenshots of Feishu tasks.

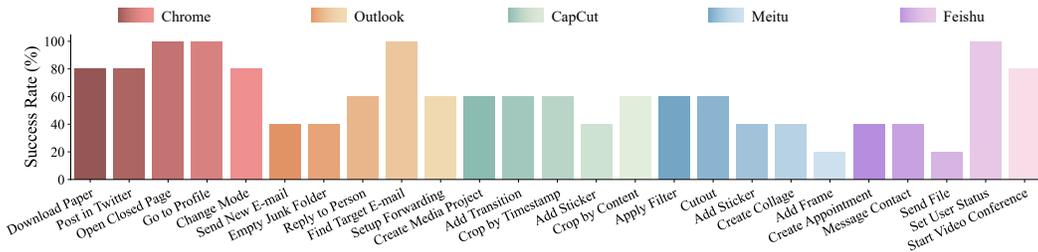


Figure 43: Success rates for tasks in software applications

I.3 QUANTITATIVE EVALUATION

We calculate **CRADLE**'s performance over the 25 tasks in the applications set. Each task is executed five times and performance is measured in three metrics: success rate, average number of steps taken by the agent (and variance over the five runs), and efficiency. *Efficiency* is defined as the ratio between the expected number of steps in a given task and the total number of steps taken by the agent. The expected number of steps per task is calculated by having humans perform each task.

Table 15 and Figure 43 show the details of the evaluation. **CRADLE** presents overall good performance over the diverse tasks and applications (compared to Expected Steps, **CRADLE** achieves an overall efficiency of 50%). However, performance for certain tasks can vary considerably due to different factors. The main reason for the higher number of task step during agent execution is the frequent incorrect positioning decisions for the mouse, *i.e.*, the backbone model chooses a position of bounding box tag that does not correspond to the UI item described in the model reasoning. We discuss examples of task-specific issues in Sections I.5 and I.6 below.

It is worth noting that in Chrome's task 3 ("Open the last closed page"), **CRADLE** knows how to use the shortcut key directly, calling the `key_press` skill directly with the correct keyboard shortcut: `'Ctrl + Shift + T'`, whereas humans typically do not know this.

To further evaluate the performance of **CRADLE** in diverse software applications scenarios, we provide quantitative results over OSWorld, a new contemporaneous benchmark with similar characteristics to our settings. More details in Appendix J and overview of the results in Table 16.

I.4 IMPLEMENTATION DETAILS

The implementation of **CRADLE** targeting all five software applications follows the GCC setting and framework modules (which include Information Gathering, Self-Reflection, Task Inference, Skill Curation, Action Planning, and Action Execution). Implementation details of the overall framework

3024 Table 15: Application Software results. *Success Rate* determines the ratio of successful completions over five
 3025 runs. *Average Steps* refers to the number of actions the agent takes to fulfil a task, if successful. *Expected Steps*
 3026 represents the number of steps as estimated by humans performing the task. *Efficiency* represents the ratio
 3027 between the expected number of steps and the total number of steps taken by the agent.

3028	Software	Success Rate	Average Steps	Expected Steps	Efficiency
3029	Chrome	88%	8.23 ± 6.75	4.20	48.05%
3030	Download Paper	80%	16.00 ± 5.52	6	37.50%
3031	Post in Twitter	80%	11.75 ± 5.26	7	61.14%
3032	Open Closed Page	100%	1.00 ± 0	3	300.00%
3033	Go to Profile	100%	4.00 ± 0.63	1	25.00%
3034	Change Mode	80%	11.25 ± 4.71	4	35.56%
3035	Outlook	60%	7.13 ± 5.61	4	48.48%
3036	Send New E-mail	40%	11.00 ± 4	5	45.45%
3037	Empty Junk Folder	40%	8.50 ± 3.50	3	35.29%
3038	Reply to Person	60%	8.33 ± 4.71	4	48.02%
3039	Find Target E-mail	100%	1.40 ± 0.80	1	71.43%
3040	Setup forwarding	60%	12.00 ± 4.90	7	58.33%
3041	CapCut	56%	10.87 ± 5.56	4.80	44.16%
3042	Create Media Project	60%	13.67 ± 5.25	7	51.20%
3043	Add transition	60%	10.67 ± 4.03	4	37.49%
3044	Crop by Timestamp	60%	11.00 ± 5.66	5	45.45%
3045	Add Sticker	40%	12.00 ± 8.00	4	33.33%
3046	Crop by Content	60%	7.00 ± 1.41	4	57.14%
3047	Meitu	44%	12.36 ± 3.34	8.00	64%
3048	Apply Filter	60%	14.67 ± 2.36	7	47.72%
3049	Cutout	60%	9.33 ± 1.89	5	53.59%
3050	Add Sticker	40%	9.50 ± 0.50	8	84.21%
3051	Create Collage	40%	16.00 ± 2.00	12	75.00%
3052	Add Frame	20%	13.00 ± 0.00	7	53.85%
3053	Feishu	56%	7.50 ± 4.50	4.00	46.07%
3054	Create Appointment	40%	8.00 ± 1.00	4	50.00%
3055	Message Contact	40%	6.00 ± 1.00	3	50.00%
3056	Send file	20%	11.00 ± 0.00	7	63.64%
3057	Set User Status	100%	14.60 ± 7.50	3	20.55%
3058	Start Video Conference	80%	4.50 ± 2.60	3	46.15%

3062 are described in Appendix D. Therefore, here we emphasize any application-specific differences or
 3063 customization.
 3064

3065 To apply CRADLE to the target application set described in this appendix, we start with base com-
 3066 mon prompts, and customize those prompts for specific modules, if necessary, to handle application-
 3067 specific characteristics. For example, for CapCut we add few-shot examples for Self-Reflection, to
 3068 let it properly perform success detection, as the application UI by itself is non-standard and some-
 3069 times provides little post-action feedback to users, making it harder for the backend model to deter-
 3070 mine action success.

3071 **Information Gathering.** Noticeably, GPT-4o presents the same limitations in both spatial reasoning
 3072 (*e.g.*, confusing up/down, left/right) and image understanding identifying specific UI items or the
 3073 state of the forefront GUI, across all applications.
 3074

3075 To help mitigate such issues, we perform augmentation on the captured screenshots similarly to the
 3076 Set-of-Mark (SoM) approach Yang et al. (2023a), by only utilizing SAM Kirillov et al. (2023) to
 3077 generate potential UI items bounding boxes and assign them numerical tags. Our SoM-like augmen-
 3078 tation differs from recent agent-related work (*e.g.*, (Zhang et al., 2024; Xie et al., 2024)), which use

OS-specific APIs to draw ground-truth bounding boxes for interactable elements (plus UI structure info, like types and element tree) to the results, while **CRADLE** relies only on image input and the segmentation output as augmentation. To make this distinction explicit, we call our augmentation approach SAM2SOM¹⁰. Figure 47 illustrates the difference. While our approach produces many more potential bounding boxes, it is more general by relying only on a screenshot (or video frame).

To ensure all bounding box labels are consistently positioned, **CRADLE**'s SAM2SOM implements two rendering styles, as shown in Figure 45 first and second rows. In the *standard* style, we pad the SAM2SOM-enhanced image when showing the label IDs in the upper left corner of the bounding boxes (to prevent labels from hiding the contents of small areas), so no numerical label ID is drawn outside the image area). In the *uniform* style, all bounding boxes utilize single-color borders with labels in black text over white background, placed within the bounding box area (top left corner).

Moreover, in specific situations we may still need to refine SAM2SOM's output further. For example, in the Feishu case, we observe that watermarks generated by the software affect the segmentation negatively, complicating GPT-4o's selection of the correct bounding boxes to interact with. Therefore, we implement a simple filtering method for such watermarks. This filter is enabled only in the Feishu benchmark and, as shown in Figure 46, can greatly reduce the number of unnecessary bounding boxes (from 216 to 166, in this example).

In addition to using the SAM2SOM method for image augmentation, we also redraw the mouse pointer not present in captured screenshots in a more prominent magenta color based on its screen position, to emphasize both its presence and position for image understanding (e.g., Figure 44). The augmentation process in Information Gathering can then result in four versions of a screenshot: a) base image, b) SAM2SOM image, c) base image with mouse pointer, and d) SAM2SOM image with mouse pointer.

Self-Reflection. As the applications in the software set are much less dynamic than complex games, there is no need to send multiple video frames to Self-Reflection. For the software applications, pre- and post-action screenshot usually suffice, i.e., one image before and one image after an action is executed. Digital games often have continuous and dynamic environments that require multiple frames to properly capture the full context and thus help the backbone LMMs understand what happened. In contrast, software operations are typically more discrete and static, where the state before and after an action provides sufficient information for most analysis.

Nonetheless, we find that irrespective of images used, GPT-4o sometimes can have difficulty determining the success of certain tasks. For example, when downloading a file on Chrome, after either pressing 'Ctrl + S', or using a 'Save' menu, the agent must also press 'Enter' or click the 'Save' button to complete the task. However, GPT-4o often assumes the task is complete when the dialog opens and before this final step. Similar cases of incorrect conclusion happen when an action correctly closes a new panel or dialog. To address this category of issues, we add mandatory reasoning rules in the prompt for the Self-Reflection module to help mitigate such mistakes. If for specific applications this still remains an issue, we can use few-shot image examples to reinforce how the backend model should correctly judge success.

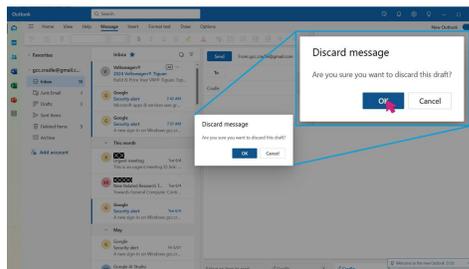


Figure 44: Sample augmented image w/ drawn mouse pointer. Zoom overlay shows the image difference.

¹⁰We do not claim the method itself as a core contribution. SAM2SOM is used to illustrate a possible extra capability of the backend model, as mitigation for current spatial reasoning issues.

3132
 3133
 3134
 3135
 3136
 3137
 3138
 3139
 3140
 3141
 3142
 3143
 3144
 3145
 3146
 3147
 3148
 3149
 3150
 3151
 3152
 3153
 3154
 3155
 3156
 3157
 3158
 3159
 3160
 3161
 3162
 3163
 3164
 3165
 3166
 3167
 3168
 3169
 3170
 3171
 3172
 3173
 3174
 3175
 3176
 3177
 3178
 3179
 3180
 3181
 3182
 3183
 3184
 3185

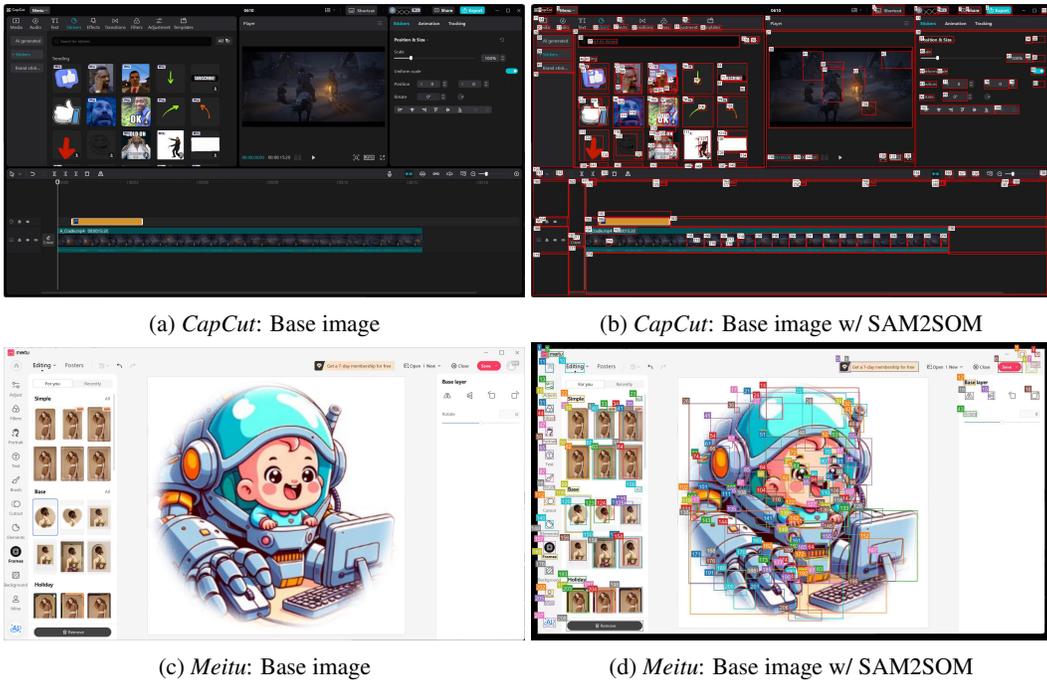


Figure 45: Image examples of the two SAM2SOM augmentation styles. As CapCut’s UI (top row) has very dark background, we utilize single-color borders with IDs in black text over white background, placed within the bounding box area. Other application software and OSWorld use the "standard" SAM2SOM multi-color style, as shown for Meitu (bottom row).

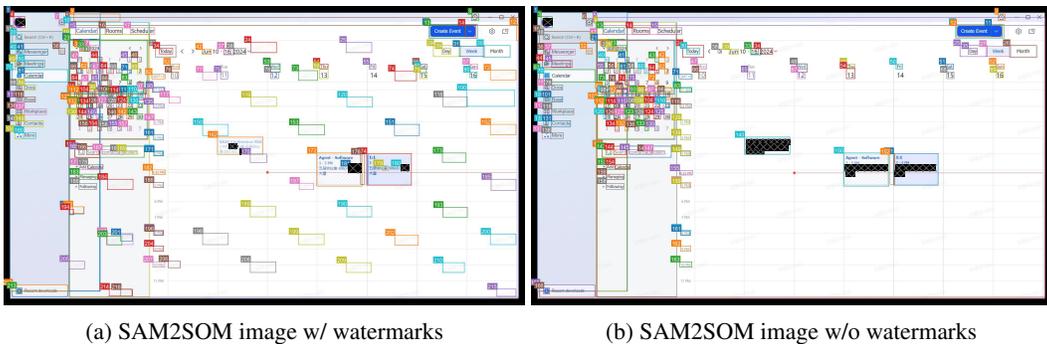


Figure 46: Examples of filtering watermark in Feishu. The number of labels is greatly reduced from 216 to 166.

3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239

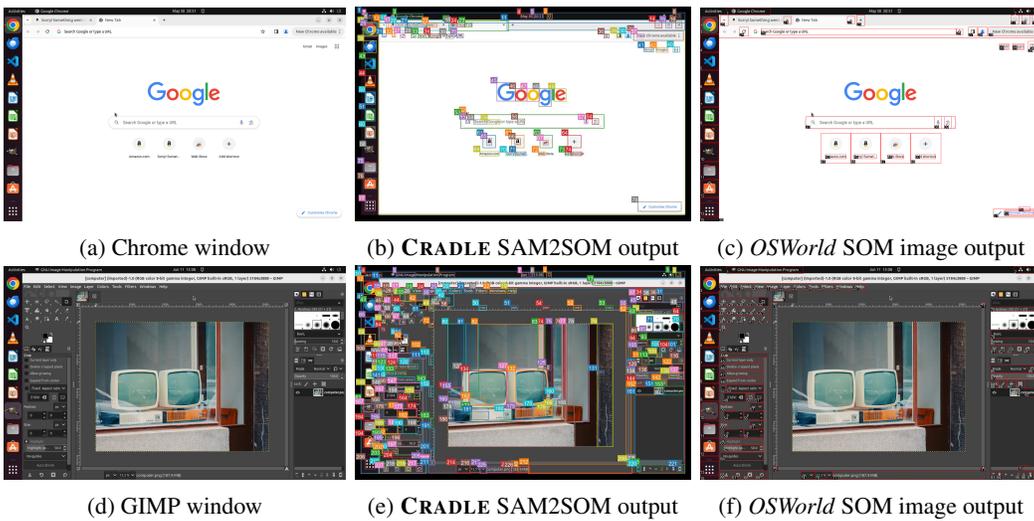


Figure 47: Comparison of **CRADLE**'s visual-only SAM2SOM and *OSWorld*'s API-based SOM image results. Chrome: 78 vs. 53 bounding boxes; GIMP: 227 vs. 98 bounding boxes.

3240 **Skill Curation.** In software tasks, direct skill
 3241 generation was not necessary, as UI operations
 3242 generally map closely to specific mouse or key-
 3243 board actions, making them more straightfor-
 3244 ward. In contrast, digital game environments
 3245 involve continuous interactions and decision-
 3246 making, raising new previously undiscovered
 3247 information, and requiring the development of
 3248 new skills to handle novel scenarios and adapt
 3249 to changing contexts.

3250 However, we do add some additional pre-
 3251 defined skills, on a per-application basis, for
 3252 specific knowledge like less-widely known key-
 3253 board shortcuts which could be learnt from the
 3254 application. For example, CapCut’s shortcuts
 3255 screen, shown in Figure 48, or toolbar/icon
 3256 processing output similarly to the process de-
 3257 scribed for Cities: Skylines. Moreover, we also
 3258 introduce pre-defined complex skills to demon-
 3259 strate **CRADLE**’s capability to leverage tools into
 3260 novel functionality, such as using GPT-4o as a tool to extract information from a video to complete
 3261 task 5 in CapCut.

3262 When dealing with shortcuts, *e.g.*, as alternatives to mouse operations, it may be the case that specific
 3263 shortcuts require "calibration". For example, using the keyboard to navigate the timeline in CapCut
 3264 (as seen in the bottom area of Figure 45b) requires mapping the keyboard shortcut ('Alt + arrow
 3265 keys') to pixels or time, which we perform a priori and use the mapping in the pre-defined skill
 3266 `go_to_timestamp(seconds)`.

3267 **Task Inference.** During the execution of an application task, we let GPT-4o decompose the execu-
 3268 tion strategy for the next step based on the overall task description and the subtask description. If
 3269 the previous task decomposition is found to be unreasonable, a new decomposition plan should be
 3270 proposed and this is evaluated at each iteration round.

3271 **Action Planning.** To enable usage of SAM2SOM, for Action Planning, we insert new mouse skills,
 3272 which mirror existing coordinates-based mouse skills (*i.e.*, that use x,y coordinates), but take a
 3273 bounding box numerical label as an argument.

3274 Furthermore, unlike in game playing, which focuses on performing one action per turn, when ma-
 3275 nipulating software **CRADLE** can be configured to perform two actions in sequence and thus lower
 3276 interaction frequency requirements to the backend model. We find that GPT4-o can usually correctly
 3277 output two-step compound actions. For example, when performing a search in the browser, it can
 3278 typically output two consecutive action steps, *e.g.*, `type_text(text='{user_query}')`, followed by the
 3279 required `press_key(key='enter')`.

3280 **Action Execution.** While atomic and composite skills can involve complex operations, Action
 3281 Execution happens over the regular **CRADLE** action space, as shown in Table 7. For example,
 3282 during Action Execution, a post-processing step converts the bounding box calls into regular mouse
 3283 actions, using the centroid of a given bounding box as its coordinates for regular mouse operations.

3284 Tool usage, like calling GPT-4o separately to analyze the contents of a media file, is not considered
 3285 as an action, as tools do not operate on the environment, only as code steps inside a composite skill.

3287 I.5 CASE STUDIES

3289 I.5.1 TASK HARDNESS

3290 It is well known that the difficulty of task completion can vary widely between humans and agents.
 3291 The results in Table 15 help illustrate some such cases. While many application operation issues
 3292 may be attributed to UI variety or non-conformity, that is not necessarily the main source of task
 3293 hardness (*i.e.*, how unexpectedly complex performing an operation is).

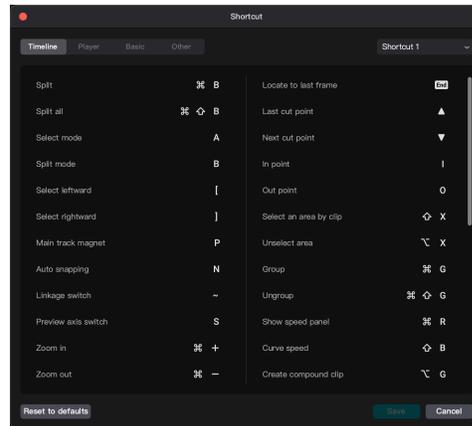


Figure 48: Shortcuts screen in CapCut.

3294 Here we use Outlook, a well-known e-mail client, as a case study to discuss how different factors
3295 affect **CRADLE** task completion in real-world application situations (the exact version used is listed
3296 in Table 12). Taking task 1 ("Create a new e-mail to {email_address} with the subject 'Hello friend'
3297 and send it.") as an example, a success rate of 40% and efficiency of 45.45% may seem lower than
3298 expected.

3299 Such a task could be reasonably broken down into steps like: a) Create new e-mail, b) Add recipient,
3300 c) Write title, and d) Send e-mail. And the Task Inference module performs such decomposition
3301 consistently. However, Action Planning needs to define specific actionable operations with mouse
3302 and keyboard to execute each step.

3303 Firstly, **CRADLE** needs to decide based on the knowledge and visual understanding capabilities
3304 available to it to either use a known keyboard shortcut (*e.g.*, 'Ctrl + N') or to click at the "New mail"
3305 button. In our experiments, **CRADLE** tends to chose clicking on the button, which is then affected by
3306 the previously discussed issues that led to the integration of SAM2SOM into the framework. Issues
3307 in spatial reasoning issues or icon/image understanding may cause a few incorrect click attempts.
3308

3309 Adding the recipient to the e-mail requires typing an address at the appropriate location, *i.e.*, the
3310 typical "To" field. This can be accomplished in multiple ways, mainly by typing the address on the
3311 UI next to the "To" item or choosing a pre-existing contact.

3312 Clicking on the "To" button triggers the UI to search and select a pre-existing contact e-mail address
3313 (with no option of adding a new contact entry, which requires first accessing the "Contacts" menu,
3314 outside of "Mail"). Moreover, the UI interaction sequence to select an existing contact can be un-
3315 intuitive even to experienced users, requiring a minimum of four steps, at each step offering multiple
3316 UI options that go away from contact selection. Attempting this flow usually leads **CRADLE** to
3317 exceed the maximum number of allowed step as it gets confused by the UI design.

3318 Nonetheless, choosing the simpler alternative of typing the e-mail address (assuming the correct text
3319 field is selected) triggers assistive UI pop-ups (as shown in Figure 50), which lead GPT-4o to falsely
3320 conclude the e-mail address is either already typed at the correct location or that it is duplicated
3321 and needs to be edited/removed. Furthermore, the pop-ups partially hide the subject area, making it
3322 harder for **CRADLE** to choose the next UI item to interact with for the next task step.

3323 Similar issues with positioning and correctly identifying the typed subject text can also occur, but at
3324 a much smaller frequency.

3325 Lastly, completing the task and sending the e-mail requires step similar to creating a new message.
3326 But determining send success requires additional attention/reflection as not all cases of the "Send
3327 mail" interface disappearing indicate a successful send (*e.g.*, clicking on an unrelated e-mail on the
3328 Inbox or closing the current window pop-up).

3329 The Self-Reflection module plays a key role in moving task completion forward by detecting failed
3330 attempts at executing each sub-task and providing rationale for failures, even if Information Gath-
3331 ering and Action Planning make repeated mistakes. Such feedback from Self-Reflection and allows
3332 Action Planning to tune its process and move ahead.
3333

3334

3335 I.5.2 TOOL USE IN CAPCUT

3336
3337 Some general computer control tasks may require additional capabilities during execution prepara-
3338 tion that can benefit from external tools to enhance agent abilities.
3339

3340 When performing video editing, like in CapCut, a user may need to determine the precise frames
3341 to operate on based on video content. For such scenarios, **CRADLE** can easily leverage tool-using
3342 skills, like the LMM's ability to understand actions in a sequence of video frames, enabling it to
3343 comprehend video content and identify the exact frames for editing.

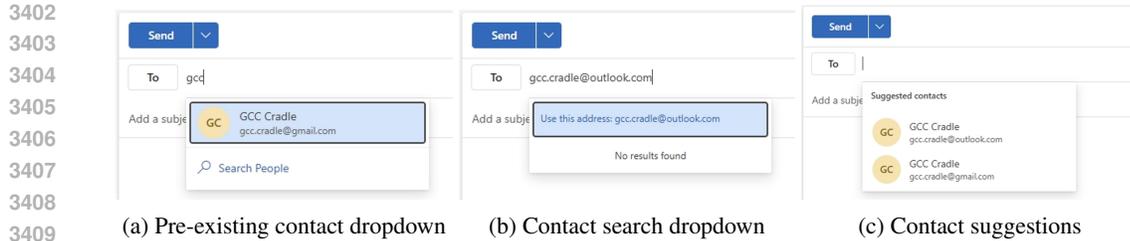
3344 We exemplify such tasks with task 5 ("Crop the video when the ball enters the goal, and then
3345 export the video") for CapCut, as illustrated in Figure 49. This means our agent can effectively
3346 execute tool usage to find the specific frame where "the ball enters the goal". After the first
3347 round of Task Inference, **CRADLE** decomposes the task into three subtasks: 1. Identify the ex-
act frame, 2. Crop the video, and 3. Export the video. Action Planning can then plan to execute

3348 'get_information_from_video(event)' from our curated skills and generate "ball enters the goal" as
 3349 its required argument for execution.

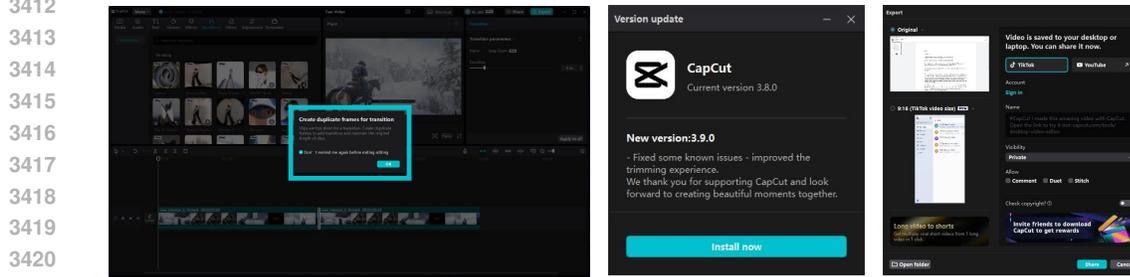
3350 In this skill, we input a frame set of the video at 1 fps to identify the specific frame where the event
 3351 occurs. The response is then recorded in Episodic Memory to ensure that subsequent operations can
 3352 accurately utilize it and target the moment when the action occurs. Across subsequent iterations,
 3353 **CRADLE** can then correctly plan and execute the remaining necessary actions for task completion:
 3354 'go_to_timestamp(seconds=8)', 'delete_right()', and 'export_project()'.
 3355

3356 We have integrated few-shot learning into Self-Reflection to ensure **CRADLE** recognizes that follow-
 3357 ing export_project(), the expected screen is the CapCut application main window. This information
 3358 allows it to verify the successful execution of the task, leading to success detection for the overall
 3359 task.





3410 Figure 50: Visual behaviour in Outlook that may lead GPT-4o to visual understanding mistakes.



3422 Figure 51: Different *CapCut* pop-ups

3423
3424
3425
3426
3427
3428
3429
3430
3431
3432

CapCut is affected by both factors, as further illustrated in Figure 51. Moreover, its UI includes non-standard layouts involving precise positioning and drag/dropping. Lack of such prior knowledge by GPT4-o and differences in behaviour between similar functions, may also lead to mistakes in trying to decompose actions to perform. E.g., "Add an effect" requires very different UI-interaction depending on details. Users can add effects in three different ways: i) dragging an effect to the timeline; ii) click the plus sign in a given effect in the effects panel, which adds the effect to the current place on the timeline; and iii) drag an effect directly onto a video and apply the effect to the entire video.

3433 Visual Context Detail.

3434
3435
3436

GPT-4o still struggles with detailed visual understanding and over-relies on textual information or hallucinations, which results in insufficient attention to visual context and leads to understanding and reasoning mistakes.

3437
3438
3439
3440
3441

One such common example is GPT-4o declaring a dialog state to be ready to press a button like "Save", while ignoring no file name was provided, even if GPT-4o has been prompted to check for such situations. The same applies to it suggesting keyboard shortcuts to open menus that do not exist in the image being interpreted, e.g., trying to press 'Alt + F' to open the "File" menu on a screenshot that has no "File" menu.

3442
3443
3444
3445

Lastly, this lack of attention to context details can also affect understanding the outcome of operations over visual content, leading to incorrect estimation of operation success, e.g., when retouching an image or deciding between a circle and a heart for a shape form.

3446 J OSWORLD

3447 3448 J.1 INTRODUCTION TO OSWORLD

3449
3450
3451
3452

OSWorld is a scalable, computer environment designed for multimodal agents. This platform provides a unified environment for assessing open-ended computer tasks involving various applications.

3453 J.2 OSWORLD TASKS

3454
3455

OSWorld is a benchmark suite of 369 real-world computer tasks (mostly on an Ubuntu Linux environment, but including a smaller set on Microsoft Windows) collected from authors and diverse

3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509

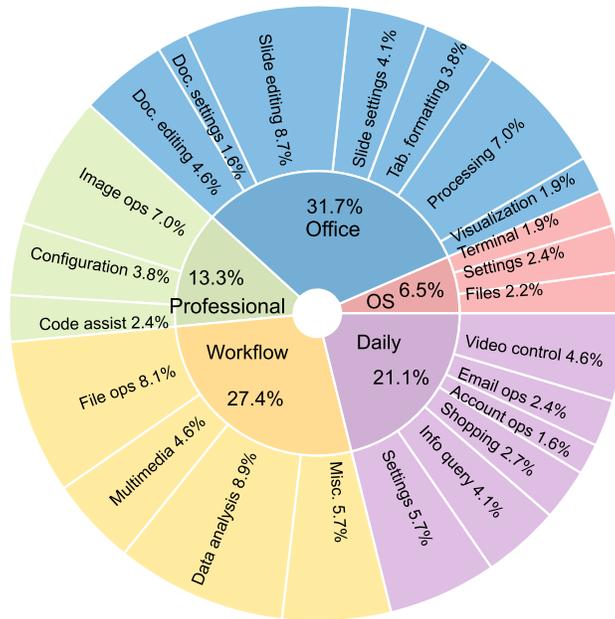


Figure 52: Task instructions distribution in OSWorld Xie et al. (2024)

sources such as forums, tutorials, guidelines. Each task is annotated with a natural language instruction and a manually crafted evaluation script for scoring.

J.3 IMPLEMENTATION DETAILS

The OSWorld environment uses a virtual machine that takes in Python scripts based on PyAutoGUI for actions and provides screenshots and an accessibility tree for observations. We strictly follow the GCC settings. Our agent only uses the screenshot as input and outputs Python scripts using PyAutoGUI methods to control the keyboard and mouse (these operations are analogous to the regular action space for CRADLE). All 369 tasks use a same set of prompt templates.

We employ GPT-4o as the framework’s backbone model. We use the default experimental settings, as in OSWorld’s baseline agent. The executable action space is the same as the OSWorld setting, the atomic skills are as follows:

- **Mouse Actions**

- *move_mouse_to_position(x, y)*: Moves the mouse to a specified position on the screen.
- *click_at_position(x, y)*: Performs a click at a specified position.
- *mouse_down(button)*: Presses the specified mouse button.
- *mouse_up(button)*: Releases the specified mouse button.
- *right_click(x, y)*: Right-clicks at the specified position.
- *double_click_at_position(x, y)*: Double-clicks at the specified position.
- *mouse_drag(x, y)*: Drags the cursor to the position.
- *scroll(direction, amount)*: Scrolls the mouse wheel up or down by a specified amount.

- **Keyboard Actions**

- *type_text(text)*: Types the specified text.
- *press_key(key)*: Presses and releases the specified key.
- *key_down(key)*: Holds a specified key.
- *key_up(key)*: Releases a specified key.
- *press_hotkey(keys)*: Presses a combination of keys and releases them in the opposite order (e.g., Ctrl+C), useful for shortcuts.

• Task Status

- *task_is_not_feasible()*: Indicates that the task cannot be completed, providing feedback for scenarios where the agent encounters infeasible tasks.

Many of these basic skills require GPT-4o to directly output an (x,y) position based on a screenshot. Given that the current GPT-4o is not able to achieve such precise control, we use a grounding tool to augment the screenshot. This way, GPT-4o only needs to choose an object ID. With the object ID and the bounding box of the object, we automatically convert it to the (x,y) position needed for skill execution. Instead of having GPT-4o directly choose the executable skills that require (x,y) position input, we provide several skills that only require a label ID as input for GPT-4o.

• Actions with Grounding Tools

- *click_on_label(label_id)*: Clicks on a specified label in the grounding result.
- *double_click_on_label(label_id)*: Double-clicks on a specified label in the grounding result.
- *hover_over_label(label_id)*: Moves the mouse to hover over a specified label in the grounding result.
- *mouse_drag_to_label(label_id)*: Drags the mouse to a specified label in the grounding result.

Information Gathering. Tasks in OSWorld require pixel-level mouse control. While GPT-4 exhibits grounding ability, using tools like SAM can further augment the screenshot with the grounding of icons in complex computer control tasks. The bounding box is helpful for GPT-4 to understand the occurrence of objects on the screen and can also be used to calculate the precise position for mouse control.

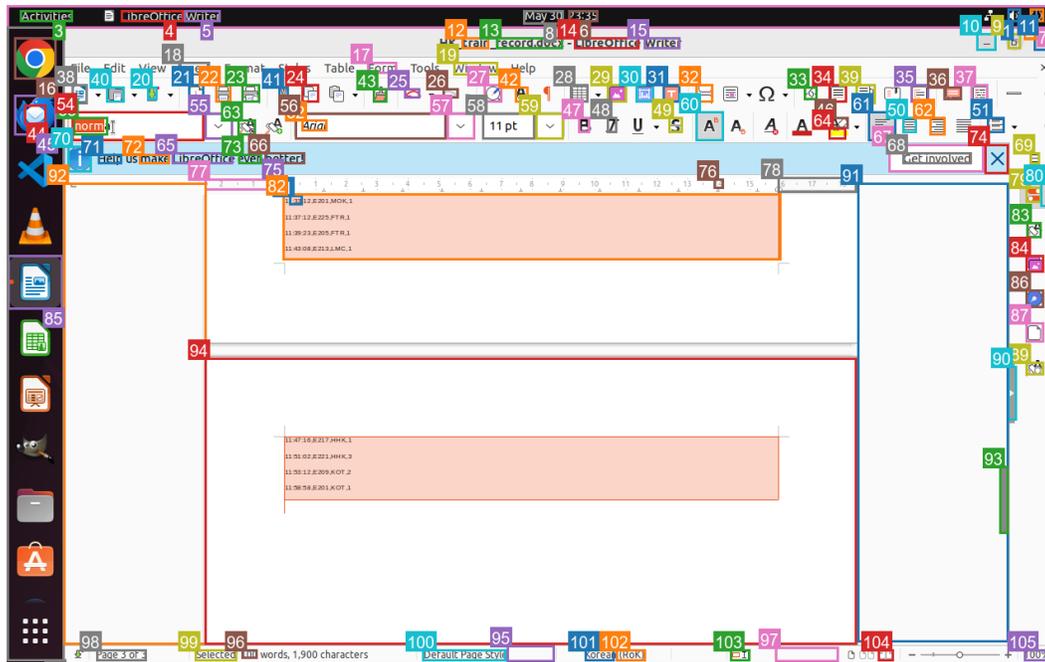


Figure 53: Augmented screenshot using CRADLE’s SAM2SOM

Self-Reflection. The reflection module evaluates whether previous actions have been successfully executed and determines if the entire task was successful. The self-reflection module is important for tasks in OSWorld, which are sequential decision-making problems that require re-planning based on the current state and previous actions. The self-reflection module also helps to identify infeasible tasks.

3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637

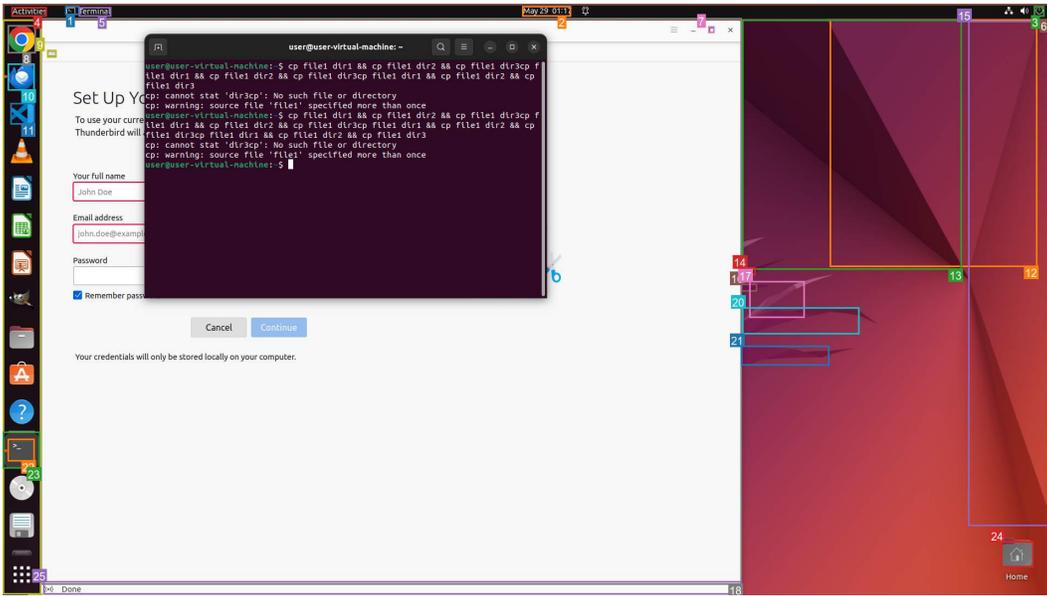


Figure 55: The agent fails to copy the files due to using incorrect commands

3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662

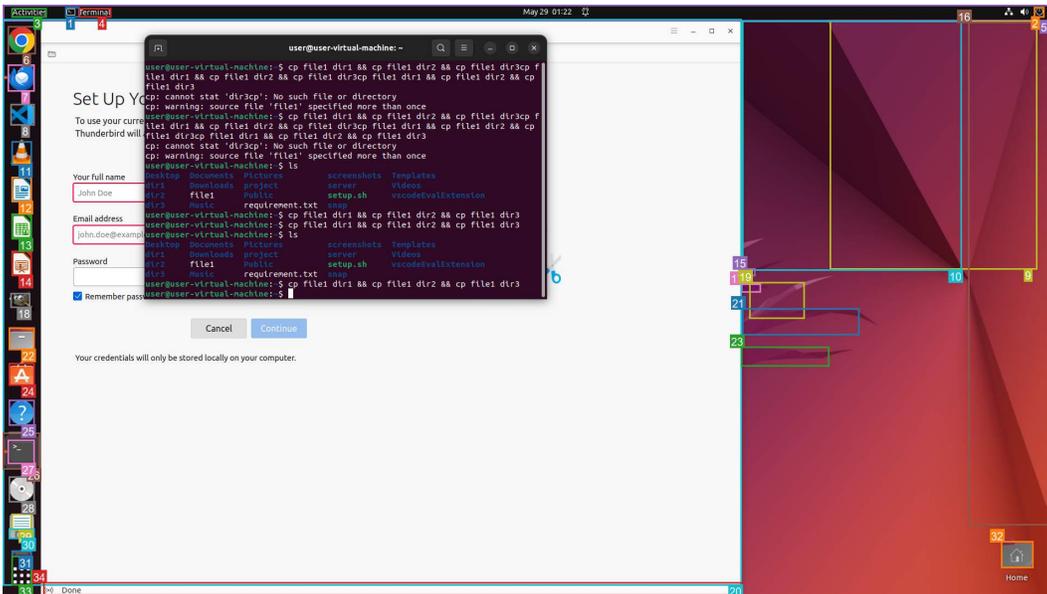
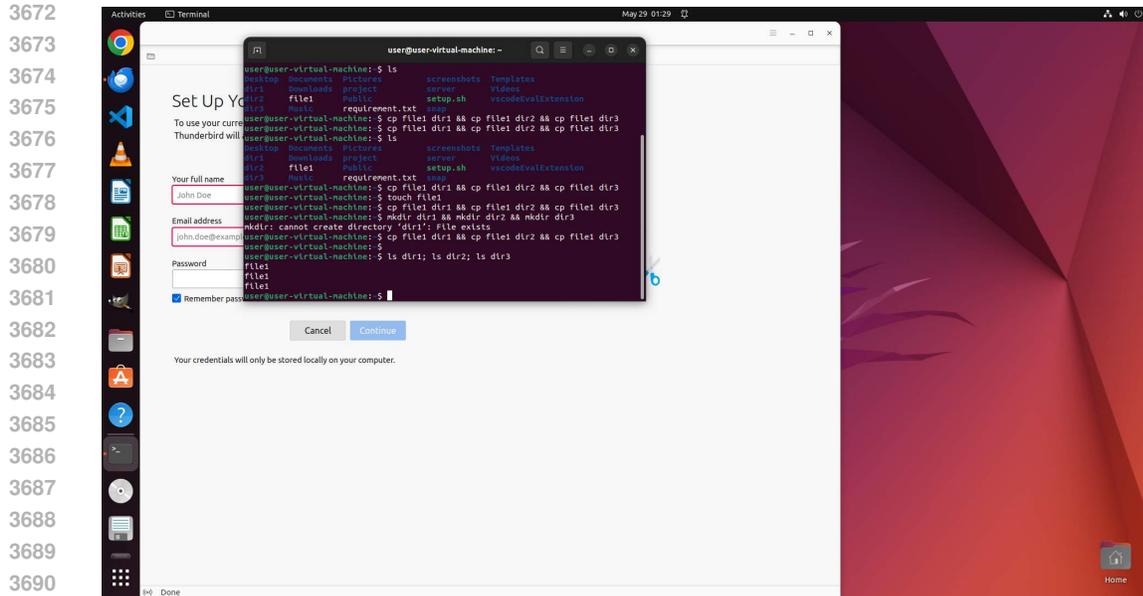


Figure 56: The agent reflects on the errors, checks the file structure and tries to debug

3663 Table 16: Detailed success rates divided by domains: OS, LibreOffice Calc, LibreOffice Impress, LibreOffice
3664 Writer, Chrome, VLC Player, Thunderbird, VS Code, GIMP, and Workflow (*i.e.*, involves multiple applica-
3665 tions).

Method	OS (24)	Calc (47)	Impress (47)	Writer (23)	VLC (17)	TB (15)	Chrome (46)	VSC (23)	GIMP (26)	Workflow (101)
GPT-4o	8.33	0.00	6.77	4.35	16.10	0.00	4.35	4.35	3.85	5.58
GPT-4o+SoM	20.83	0.00	6.77	4.35	6.53	0.00	4.35	4.35	0.00	3.60
CRADLE	16.67	0.00	4.65	8.70	6.53	0.00	8.70	0.00	38.46	5.48

3671



3692 Figure 57: The agent checks if the files have already been copied

3693 J.6 QUANTITATIVE EVALUATION

3694 The detailed success rates for each application are listed in Table 16. We followed the same experi-
3695 mental settings as the OSWorld paper, running the experiment only once. Our results show that our
3696 agent performs better in the Chrome and GIMP domains. However, the difference in performance in
3697 the OS, Writer, and VSC domains is less statistically significant due to the smaller number of tasks.
3698 While improved information gathering and self-reflection empowered the agent in these domains,
3699 the complex pipeline and limitations of current grounding tools and GPT-4 hindered performance in
3700 domains like VLC and VSC. We identify these limitations as future directions for implementing the
3701 agent in real-world scenarios.

3702 K CRADLE PROMPTS

3703 Here we exemplify the utilized prompts, for each module in the framework. All prompts and cus-
3704 tomizations are included in the relevant branch in CRADLE’s open-source repository in GitHub ¹¹.

3705 K.1 PROMPTS FOR RDR2

3706 Prompt 1: RDR2: Information Gathering prompt.

3707 Assume you are a helpful AI assistant integrated with ‘Red Dead
3708 Redemption 2’ on the PC, equipped to handle a wide range of tasks in
3709 the game. Your advanced capabilities enable you to process and
3710 interpret gameplay screenshots and other relevant information.

3711 <\$few_shots\$>

3712 <\$image_introduction\$>

3713 Current task:

3714 <\$task_description\$>

3715 Target_object_name: Assume you can use an object detection model to
3716 detect the most relevant object for completing the current task if

3717 ¹¹<https://cradle2024acc.github.io/Cradle>

3726 needed. What object should be detected to complete the task based on
 3727 the current screenshot and the current task? You should obey the
 3728 following rules:

- 3729 1. The object should be relevant to the current target or the
 3730 intermediate target of the current task. Just give one name without
 3731 any modifiers.
- 3732 2. If no explicit weapon is specified on the weapon radial menu,
 3733 prioritize choosing 'gun' as the weapon.
- 3734 3. If no explicit shoot target is specified, prioritize choosing 'person'
 3735 as the target.
- 3736 4. If no explicit item is specified, only output 'null'.
- 3737 5. If the object name belongs to the person type, replace it with 'person'
 3738 '.
- 3739 6. If there is no need to detect an object, only output "null".
- 3740 7. If you are on the trade, map, inventory, or satchel interfaces, only
 3741 output 'null'.

3742 Reasoning_of_object: Why was this object chosen, or why is there no need
 3743 to detect an object?

3744 Description: Please describe the screenshot image in detail. Pay
 3745 attention to any maps in the image, if any, especially critical icons
 3746 , red paths to follow, or created waypoints. If there are multiple
 3747 images, please focus on the last one.

3748 Screen_classification: Please select the class that best describes the
 3749 screenshot among "Inventory", "Radial menu", "Satchel", "Map", "Trade
 3750 ", "Pause", and "General game interface without any menu". Output the
 3751 class of the screenshot in the output of Screen_classification.

3752 Reasoning_of_screen: Why was this class chosen for the current screenshot
 3753 ?

3754 Movement: Does the current task require the character to go somewhere?

3755 Noun_and_Verb: The number of nouns and verbs in the current task.

3756

3757 Task_horizon: Please judge the horizon of the current task, i.e., whether
 3758 this task needs multiple or only one interaction.

3759 There are two horizon types: long-horizon and short-horizon. For long-
 3760 horizon tasks, the output should be 1. For short-horizon tasks, the
 3761 output should be 0. You should obey the following rules:

- 3762 1. If the task contains only nouns without verbs, it is short-horizon.
- 3763 2. If the task contains more than one verb, it is long-horizon.
- 3764 3. If the task requires the character to go somewhere, it is long-horizon
 3765 .

3766 Short-horizon tasks are sub-goals during a long-horizon task, which only
 3767 need one interaction. There are some examples of short-horizon tasks:

- 3768 1. Pick up something: To complete this task, the character needs to
 3769 execute the action "pick up" only once, so it is short-horizon.
- 3770 2. Use or press [B] key: The character needs to press the key [B] only
 3771 once to talk, so it is short-horizon.
- 3772 3. Talk to somebody: The character needs to press a certain button once
 3773 to complete this task, so it is short-horizon.

3774 Long-horizon tasks are long-term goals, which usually need many
 3775 interactions. There are some examples of long-horizon tasks.

- 3776 1. Go outside: The character should go outside step by step, so it is
 3777 long-horizon.
- 3778 2. Approach something: The character should move closer to the target
 3779 step by step, so it is long-horizon.
- 3780 3. Keep away from something, shoot, take down, or battle with something:
 The character must engage in a series of interactions, so it is long-
 horizon.

Reasoning_of_task: Why do you make such a judgment of task_horizon?

3780
 3781 You should only respond in the format described below and not output
 3782 comments or other information.
 3783 Target_object_name:
 3784 Name
 3785 Reasoning_of_object:
 3786 1. ...
 3787 2. ...
 3788 ...
 3788 Description:
 3789 The image shows...
 3790 Screen_classification:
 3791 Class of the screenshot
 3791 Reasoning_of_screen:
 3792 1. ...
 3793 2. ...
 3794 ...
 3795 Movement:
 3796 Yes or No
 3796 Noun_and_Verb:
 3797 1 noun 1 verb
 3798 Task_horizon:
 3799 1
 3800 Reasoning_of_task:
 3801 1. ...
 3802 2. ...
 3803 ...

3804 **Prompt 2: RDR2: Gather Text Information prompt.**

3805 Assume you are a helpful AI assistant integrated with 'Red Dead
 3806 Redemption 2' on the PC, equipped to handle a wide range of tasks in
 3807 the game. Your advanced capabilities enable you to process and
 3808 interpret gameplay screenshots and other relevant information.
 3809
 3810 <\$image_introduction\$>
 3811
 3812 Information: List all text prompts on the screenshot from the top to the
 3813 bottom, even the text prompt is one word.
 3814
 3814 All information should be categorized into one or more kinds of <
 3815 \$information_type\$>. If you think a piece of information is both "A"
 3816 and "B" categories, you should write information in both "A" and "B"
 3817 categories. For example, "use E to drink water" could both be "Action
 3818 Guidance" and "Task Guidance" categories.
 3819
 3819 Item_status: The helpful information to the current context in the game,
 3820 such as the cash, amount of ammo, current using item, if the player
 3821 is wanted, etc. This content should be pairs of status names and
 3822 their values. For example, "cash: 100\$". If there is no on-screen
 3823 text and no item status, only output "null".
 3824
 3824 Environment_information: The information about the location, time,
 3825 weather, etc. This content should be pairs of status names and their
 3826 values. For example, "location: VALENTINE". If there is no on-screen
 3827 text and environment information, only output "null".
 3828
 3828 Notification: The game will give notifications showing the events in the
 3829 world, such as obtaining items or rewards, completing objectives, and
 3830 becoming wanted. Besides, it also contains valuable notifications of
 3831 the game's mechanisms, such as "Health is displayed in the lower
 3832 left corner". The content must be the on-screen text. If there is no
 3833 on-screen text or notification, only output "null".
 3833
 3833 Task_guidance: The content should obey the following rules:

3834 1. The content of task guidance must be an on-screen text prompt,
3835 including the menu and the general game interface.

3836 2. The game will give guidance on what should be done to proceed with the
3837 game, for example, "follow Tom". This is task guidance.

3838 3. The game will give guidance on how to perform a task using keyboard
3839 keys or mouse buttons, for example, "use E to drink water". This is
3840 task guidance.

3841 4. If no on-screen text prompt or task guidance exists, only output "null
3842 ". Never derive the task guidance from the dialogue or notifications.

3843 Action_guidance: The game will give guidance on how to perform a task
3844 using keyboard keys or mouse buttons; you must generate the code
3845 based on the on-screen text. The content of the code should obey the
3846 following code rules:

3847 1. You should first identify the exact keyboard or mouse key represented
3848 by the icon on the screenshot. 'Ent' refers to 'enter'. 'RM' refers
3849 to 'right mouse button'. 'LM' refers to 'left mouse button'. You
3850 should output the full name of the key in the code.

3851 2. You should refer to different examples strictly based on the word used
3852 to control the key, such as 'use', 'hold', 'release', 'press', and 'click'.

3853 3. If 'use' or 'press' is in the prompt to control the keyboard key or
3854 mouse button, `io_env.key_press('key', 2)` or `io_env.mouse_click('button', 2)` must be used to act on it. Refer to Examples 1, 2, and 3.

3855 4. If there are multiple keys, `io_env.key_press('key1,key2', 2)` must be
3856 used to act on it. Refer to Example 4.

3857 5. If 'hold' is in the prompt to control the keyboard key or mouse button
3858 , it means keeping the key held with `io_env.key_hold` or the button
3859 held with `io_env.mouse_hold` (usually indefinitely, with no duration).
3860 If you need to hold it briefly, specify a duration argument. Refer
3861 to Examples 5 and 6.

3862 6. All durations are set to a minimum of 2 seconds by default. You can
3863 choose a longer or shorter duration. If it should be indefinite, do
3864 not specify a duration argument.

3865 7. The name of the created function should only use phrasal verbs, verbs,
3866 nouns, or adverbs shown in the prompt and should be in the verb+noun
3867 or verb+adverb format, such as `drink_water`, `slow_down_car`, and
3868 `ride_faster`. Note that words that do not show in the prompt are
3869 prohibited.

3870 This is Example 1. If "press" is in the prompt and the text prompt on the
3871 screenshot is "press X to play the card", your output should be:

```
3872 ```python
3873 def play_card():
3874     """
3875     press "x" to play the card
3876     """
3877     io_env.key_press('x', 2)
3878     ```
```

3879 This is Example 2. If the instructions involve the mouse and the text
3880 prompt on the screenshot is "use the left mouse button to confirm",
3881 your output should be:

```
3882 ```python
3883 def confirm():
3884     """
3885     use "left mouse button" to confirm
3886     """
3887     io_env.mouse_click("left mouse button")
3888     ```
```

3889 This is Example 3. If "use" is in the prompt and the text prompt on the
3890 screenshot is "use ENTER to drink water", your output should be:

```
3891 ```python
3892 def drink_water():
3893     """
3894     use "enter" to drink water
3895     """
3896     io_env.key_press('enter', 2)
3897     ```
```

```

3888     """
3889     io_env.key_press('enter', 2)
3890     """
3891 This is Example 4. If "use" is in the prompt and the text prompt on the
3892 screenshot is "use W and J to jump the barrier", your output should
3893 be:
3894 ```python
3895 def jump_barrier():
3896     """
3897     use "w" and "j" to jump the barrier
3898     """
3899     io_env.key_press('w,j', 3)
3900     """
3901 This is Example 5. If "hold" is in the prompt and the text prompt on the
3902 screenshot is "hold H to run", your output should be:
3903 ```python
3904 def run():
3905     """
3906     hold "h" to run
3907     """
3908     io_env.key_hold('h')
3909     """
3910 This is Example 6. If the instructions involve the mouse and the text
3911 prompt on the screenshot is "hold the right mouse button to focus on
3912 the target", your output should be:
3913 ```python
3914 def focus_on_target():
3915     """
3916     hold "right mouse button" to focus
3917     """
3918     io_env.mouse_hold("right mouse button")
3919     """
3920 This is Example 7. If "release" is in the prompt and the text prompt on
3921 the screenshot is "release Q to drop the items", your output should
3922 be:
3923 ```python
3924 def drop_items():
3925     """
3926     release "q" to drop the items
3927     """
3928     io_env.key_release('q')
3929     """
3930 Dialogue: Conversations between characters in the game. This content
3931 should be in the format of "character name: dialogue". For example, "
3932 Arthur: I'm fine". If there is no on-screen text or dialogue, only
3933 output "null".
3934 Other: Other information that does not belong to the above categories. If
3935 there is no on-screen text, only output "null".
3936 Reasoning: The reasons for classification for each piece of information.
3937 If the on-screen text prompt is an instruction on how to perform a task
3938 using keyboard keys or mouse buttons, it should also be classified as
3939 action guidance and task guidance.
3940 For action guidance, which code rules should you follow based on the word
3941 used to control the key or button, such as press, hold, release, and
3942 click?
3943 The information should be in the following categories, and you should
3944 output the following content without adding any other explanation:
3945 Information:
3946 1. ...
3947 2. ...
3948 ...

```

```

3942 Reasoning:
3943 1. ...
3944 2. ...
3945 ...
3946 Item_status:
3947 Item_status is ...
3948 Environment_information:
3949 Environment information is ...
3950 Notification:
3951 Notification is ...
3952 Task_guidance:
3953 Task is ...
3954 Action_guidance:
3955 ```python
3956 Python code to execute
3957 ```
3958 ```python
3959 Python code to execute
3960 ```
3961 ...
3962 Dialogue:
3963 Dialogue is ...
3964 Other:
3965 Other information is ...

```

Prompt 3: RDR2: Self-Reflection prompt.

```

3966 Assume you are a helpful AI assistant integrated with 'Red Dead
3967 Redemption 2' on the PC, equipped to handle a wide range of tasks in
3968 the game. Your advanced capabilities enable you to process and
3969 interpret gameplay screenshots and other relevant information. Your
3970 task is to examine these inputs, interpret the in-game context, and
3971 determine whether the executed action takes effect.
3972
3973 Current task:
3974 <$task_description$>
3975
3976 Last executed action:
3977 <$previous_action$>
3978
3979 Implementation of the last executed action:
3980 <$action_code$>
3981
3982 Error report for the last executed action:
3983 <$executing_action_error$>
3984
3985 Reasoning for the last action:
3986 <$previous_reasoning$>
3987
3988 Valid action set in Python format to select the next action:
3989 <$skill_library$>
3990
3991 <$image_introduction$>
3992
3993 Reasoning: You need to answer the following questions step by step to get
3994 some reasoning based on the last action and sequential frames of the
3995 character during the execution of the last action.
3996 1. What is the last executed action not based on the sequential frames?
3997 2. Was the last executed action successful? Give reasons. You should
3998 refer to the following rules:
3999 - If the action involves moving forward, it is considered unsuccessful
4000 only when the character's position remains unchanged across
4001 sequential frames, regardless of background elements and other people
4002 .

```

3996 3. If the last action is not executed successfully, what is the most
3997 probable cause? You should give only one cause and refer to the
3998 following rules:
3999 - The reasoning for the last action could be wrong.
4000 - Not holding enough time should not be considered in this part.
4001 - If it is an interaction action, the most probable cause was that the
4002 action was unavailable or not activated at the current place.
4003 - If it is a movement action, the most probable cause was that you were
4004 blocked by seen or unseen obstacles.
4005 - If there is an error report, analyze the cause based on the report.
4006 You should only respond in the format as described below:
4007 Reasoning:
4008 1. ...
4009 2. ...
4010 3. ...
4011 ...

Prompt 4: RDR2: Task Inference prompt.

4013 Assume you are a helpful AI assistant integrated with 'Red Dead
4014 Redemption 2' on the PC, equipped to handle a wide range of tasks in
4015 the game. You will be sequentially given <\$event_count\$> screenshots
4016 and corresponding descriptions of recent events. You will also be
4017 given a summary of the history that happened before the last
4018 screenshot. You should assist in summarizing the events for future
4019 decision-making.
4020 The following are <\$event_count\$> successive screenshots and
4021 corresponding descriptions:
4022 <\$image_introduction\$>
4023
4024 The following is the summary of history that happened before the last
4025 screenshot:
4026 <\$previous_summarization\$>
4027
4028 Current task:
4029 <\$task_description\$>
4030
4031 Info_summary: Based on the above input, please make a summary from the
4032 screenshots with descriptions and the history in no less than 10
4033 sentences, following the rules below.
4034 1. Summarize the tasks from the history and the current task, with a
4035 special note on the method of crucial press operations.
4036 2. Summarize the entities and behaviors mentioned in the successive
4037 descriptions.
4038 3. If entities and behaviors in the history and screenshots are missed in
4039 the descriptions, please add them to the summarization.
4040 4. Organize the summarization as a story in order of time, including the
4041 past entities and behaviors.
4042 5. Only give descriptions; do not provide suggestions.
4043
4044 Entities_and_behaviors: Entities and behaviors which are summarized, e.g
4045 ., The entities include the player's character, the target character,
4046 and horses for both the player and the target. The behaviors consist
4047 of the player character riding horseback, following the target on
4048 horseback, and moving forward to maintain a distance behind the
4049 target.
4050
4051 The output should be in the following format:
4052 Info_summary:
4053 The summary is...
4054 Entities_and_behaviors:
4055 The summary is...

4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103

Prompt 5: RDR2: Action Planning prompt.

You are a helpful AI assistant integrated with 'Red Dead Redemption 2' on the PC, equipped to handle various tasks in the game. Your advanced capabilities enable you to process and interpret gameplay screenshots and other relevant information. By analyzing these inputs, you gain a comprehensive understanding of the current context and situation within the game. Utilizing this insight, you are tasked with identifying the most suitable in-game action to take next, given the current task. You control the game character and can execute actions from the available action set. Upon evaluating the provided information, your role is to articulate the precise action you would deploy, considering the game's present circumstances, and specify any necessary parameters for implementing that action.

Here is some helpful information to help you make the decision.

Current task:
<\$task_description\$>

Memory examples:
<\$memory_introduction\$>

<\$few_shots\$>

<\$image_introduction\$>

Last executed action:
<\$previous_action\$>

Reasoning for the last action:
<\$previous_reasoning\$>

Self-reflection for the last executed action:
<\$previous_self_reflection_reasoning\$>

Summarization of recent history:
<\$info_summary\$>

Valid action set in Python format to select the next action:
<\$skill_library\$>

Minimap information:
<\$minimap_information\$>

Based on the above information, you should first analyze the current situation and provide the reasoning for what you should do for the next step to complete the task. Then, you should output the exact action you want to execute in the game. You should respond to me with :

Reasoning: You should think step by step and provide detailed reasoning to determine the next action executed on the current state of the task. You need to answer the following questions step by step. You cannot miss the question number 13:

1. Only answer this question when the radial menu, trade, map, satchel or inventory interfaces are open. You should first describe each item in the screen line by line, from the top left and moving right. Is the target item in the current screen?
2. Only answer this question when the radial menu, trade, map, satchel or inventory interfaces are open. Which item is selected currently?

4104 3. Only answer this question when the character is visible in the
4105 screenshot of the current step. Where is the character in the
4106 screenshot of the current step?

4107 4. Where is the target in the screenshot of the current step based on
4108 the task description, on the left side or on the right side? Does it
4109 appear in the previous screenshots?

4110 5. Are there any bounding boxes with coordinates values and object
4111 labels, such as "door x = 0.5, y = 0.5", shown in the screenshot? The
4112 answer must be based only on the screenshot of the current step, not
4113 on any previous steps. If the answer is no, ignore the questions 6
4114 to 8.

4115 6. You should first describe each bounding box, from left to right.
4116 Which bounding box is more relevant to the target?

4117 7. What is the value x of the most relevant bounding box only in the
4118 current screenshot? The value is the central coordination (x,y) of
4119 the central point of the box.

4120 8. Based on the few shots and the value x, where is the relevant
4121 bounding box in the current screenshot? Clearly on the left side,
4122 slightly on the left side, in the center, slightly on the right side,
4123 or clearly on the right side?

4124 9. Only answer this question when the radial menu, trade, map,
4125 satchel or inventory interfaces are not open. Summarize the contents
4126 of recent history, mainly focusing on the historical tasks and
4127 behaviors.

4128 10. Only answer this question when the radial menu, trade, map,
4129 satchel or inventory interfaces are not open. Summarize the content
4130 of self-reflection for the last executed action, and do not be
4131 distracted by other information.

4132 11. What was the previous action? If the previous action was a turn,
4133 was it a left or a right turn? If the previous action was a movement,
4134 were you blocked?

4135 12. List conditions in action rule 12 and which condition is
4136 satisfied. Only when you do not satisfy any conditions, summarize the
4137 content of the minimap information.

4138 13. This is the most critical question. Based on the action rules and
4139 self-reflection, what should be the most suitable action in the
4140 valid action set for the next step? You should analyze the effects of
4141 the action step by step.

4142 Actions: The best action, or short sequence of actions without gaps, to
4143 execute next to progress in achieving the goal. Pay attention to the
4144 names of the available skills and to the previous skills already
4145 executed, if any. You should also pay more attention to the following
4146 action rules:

4147 1. You should output actions in Python code format and specify any
4148 necessary parameters to execute that action. If the function has
4149 parameters, you should also include their names and decide their
4150 values, like "move(duration=1)". If it does not have a parameter,
4151 just output the action, like "mount_horse()".

4152 2. Given the current situation and task, you should only choose the
4153 most suitable action from the valid action set. You cannot use
4154 actions that are not in the valid action set to control the character
4155 .

4156 3. If the target is not on the radial menu, trade, satchel or
4157 inventory interfaces, you MUST choose the skill 'view_next_page'. For
the map, ignore the skill 'view_next_page'.

4. If the minimap information exists, it may include angle
information for red points, yellow points, or yellow regions. Angle
information specifies the direction of the corresponding point or
area. A negative angle indicates the left side, while a positive
value signifies the right side. If the angle is 30, the corresponding
point or area is 30 degrees to the character's right. If the angle
is -50, the corresponding point or area is 50 degrees to the
character's left. Do not doubt the correctness of these angles; you
can refer to them when you approach these points or regions.

4158 5. When you decide to control the character to move, if the relevant
 4159 bounding box is clearly on the left side in the current screenshot,
 4160 you MUST turn left with a big degree. If the relevant bounding box is
 4161 slightly on the left side in the current screenshot, you MUST turn
 4162 left with a small degree. If the relevant bounding box is clearly on
 4163 the right side in the current screenshot, you MUST turn right with a
 4164 big degree. If the relevant bounding box is slightly on the right
 4165 side in the current screenshot, you MUST turn right with a small
 4166 degree. If the relevant bounding box is on the central side of the
 4167 current screenshot, you can choose to move forward.

4167 6. When you decide to control the character to move, if yellow
 4168 regions or yellow points exist in minimap information, they are
 4169 related to the current task or instruction. This implies that you
 4170 should approach within the yellow region or approach the yellow
 4171 points. You can refer to the corresponding angle information when
 4172 deciding to approach these regions or points. If red points exist in
 4173 the minimap information, they are also related to the current task or
 4174 instruction. This implies that you should turn towards them, and you
 4175 can also refer to the corresponding angle information.

4175 7. When you decide to control the character to move, if minimap
 4176 information does not exist, the 'theta' you use to turn MUST be more
 4177 than 10 degrees and less than 60 degrees.

4177 8. When you decide to control the character to move, if you are in a
 4178 normal road condition, the 'duration' you use to move forward should
 4179 be 1 second. If you have bad road conditions, such as snow, and grass
 4180 , that can slow you down, the 'duration' you use to move forward
 4181 should be 2 seconds.

4181 9. When you are exploring or searching a place, if you are leaving
 4182 the place, you MUST make a sharp turn to face the inside of the place
 4183 . Any values for degrees are allowed.

4183 10. If upon self-reflection you think the last action was unavailable
 4184 at the current place, you MUST move to another place.

4184 11. If upon self-reflection you think you were blocked, you MUST make
 4185 a moderate turn in the same direction as the previous turn action
 4186 and move forward, so that you can pass obstacles.

4186 12. The conditions to ignore the minimap information for decision-
 4187 making are: 1. When self-reflection implies you were blocked. 2. When
 4188 you were inside the highlighted area in the minimap. If any of the
 4189 conditions satisfied, you must ignore the minimap information for
 4190 decision-making even if it is relevant to the current task.

4190 13. When you are indoors, or the current task does not imply
 4191 following, you MUST not use the follow action.

4191 14. When you are outdoors, and the current task implies following,
 4192 you MUST use the follow action.

4192 15. If you were dead or the game failed, you MUST retry from the
 4193 checkpoint, and MUST NOT restart the mission.

4197 You should only respond in the format described below, and you should not
 4198 output comments or other information:

4199 Reasoning:

4200 1. ...
 4201 2. ...
 4202 3. ...

4203 Actions:

```
4204 ```python
4205     action(args1=x,args2=y)
4206 ```
```

4207 K.2 PROMPTS FOR CITIES: SKYLINES

4208 Prompt 6: Skylines: Information Gathering prompt.

4209 Assume you are a helpful AI assistant integrated with 'Cities: Skylines'
 4210 on the PC, equipped to handle a wide range of tasks in the game. Your

4212 advanced capabilities enable you to process and interpret gameplay
 4213 screenshots and other relevant information.
 4214
 4215 <\$image_introduction\$>
 4216
 4217 Current task:
 4218 <\$task_description\$>
 4219
 4220 Description: Please analyze and describe the screenshot image in detail
 4221 and then provide an overall image description. Pay attention to
 4222 anything related to the task. If there are specific features such as
 4223 characters or text, mention these as well.
 4224
 4225 Budget: Bank Balance is shown at the bottom of the screenshot.
 4226
 4227 Population: The population of the city is shown at the bottom of the
 4228 screenshot, next to the budget.
 4229
 4230 Error_message: If there are some in-game error messages, which are
 4231 usually in red color, such as "Space already occupied!", extract the
 4232 text, otherwise, only output "null".
 4233
 4234 Construction_information: If there is some in-game construction
 4235 information, which is usually in blue colors, such as "Construction
 4236 cost: 2500 Estimated production:0 m^3/week" and "Construction cost:
 4237 2500 Shoreline recommended", extract the text, otherwise, only output
 4238 "null".
 4239
 4240 Other: Other information that does not belong to the above categories. If
 4241 none of them applies, only output "null".
 4242
 4243 You should only respond in the format described below and not output
 4244 comments or other information.
 4245 Description:
 4246 The image shows...
 4247 Budget:
 4248 The amount of budget
 4249 Population:
 4250 The amount of population
 4251 Error_message:
 4252 The text of the error message
 4253 Construction_information:
 4254 The text of the construction information
 4255 Other:
 4256 Other information is

Prompt 7: Skylines: Self-Reflection prompt.

4252 Assume you are a helpful AI assistant integrated with 'Cities: Skylines'
 4253 on the PC, equipped to handle a wide range of tasks in the game. Your
 4254 advanced capabilities enable you to process and interpret gameplay
 4255 screenshots and other relevant information. Your task is to examine
 4256 these inputs, interpret the in-game context, and determine whether
 4257 the executed action takes effect.
 4258
 4259 Target task:
 4260 <\$task_description\$>
 4261
 4262 Current subtask for completing the target task:
 4263 <\$subtask_description\$>
 4264
 4265 Current coordinates:
 <\$coordinates\$>
 Last executed action for completing the subtask:

4266 <\$actions\$>
4267
4268 Error message for the last executed action:
4269 <\$error_message\$>
4270
4271 Construction information:
4272 <\$construction_information\$>
4273
4274 Summarization of recent history:
4275 <\$history_summary\$>
4276
4277 <\$image_introduction\$>
4278
4279 Reasoning: You MUST answer the following questions step by step to get
4280 some reasoning based on the last action and sequential frames during
4281 the execution of the last action.
4282 1. What is the executed action? Please answer this question not based on
4283 the sequential frames.
4284 2. Is the construction information provided in the information shown
4285 above? If yes, what is it?
4286 3. Was the last executed action successful? Give reasons. You should
4287 refer to the following rules:
4288 - Buildings and roads cannot be built on the river.
4289 - Water pumping station and water drain pipe need to be built as close as
4290 possible to the river.
4291 - If you are try_place a water pumping station and the construction
4292 information provided above shows that the estimated production is 0 m
4293 ^3/week, then it means that it is not close enough to the river. So
4294 you need to try_place to place the building to another place. If the
4295 estimated production is not 0 m^3/week, or the construction
4296 information is not provided, regard this action as a success. You
4297 should only refer to the textual construction information instead of
4298 extracting it from the sequential frames.
4299 - If you are try_place a water drain pipe and the construction
4300 information shows that shoreline is recommended. Then it means that
4301 it is not close enough to the river. So you need to try_place to
4302 place the building in another place.
4303 - Roads are prohibited from crossing together and do not build roads on
4304 water.
4305 4. If the last action is not executed successfully, what is the most
4306 probable cause? How to improve this action? You should give only one
4307 cause and refer to the following rules:
4308 - The reasoning for the last action could be wrong.
4309 - If there is an error message for the last executed action provided in
4310 the above information, analyze the cause based on the report,
4311 otherwise, you should regard that there are no error messages. You
4312 are not allowed to guess the error message by yourself.
4313 5. Is the subtask completed? Give your reasons. You MUST remember that
4314 action starts with "try_place" can NEVER complete the subtask. Only "
4315 confirm_placement()" can make the building happen and complete the
4316 task. If you want to make any confirmation, regard it as a success.
4317 6. Do you think the subtask is reasonable? Give your reasons.
4318
4319 Success: You need to output whether the last action was executed
4320 successfully or not.
4321 - If the last action is successful, you should only output 'True'.
4322 Otherwise, you should only output 'False'.
4323
4324 You should only respond in the format described below.
4325 Reasoning:
4326 1. ...
4327 2. ...
4328 3. ...
4329 4. ...
4330 5. ...

4320 6. ...
 4321 ...
 4322 Success:
 4323 True
 4324 ...

4326 **Prompt 8: Skylines: Task Inference prompt.**

4328 Assume you are a helpful AI assistant integrated with 'Cities: Skylines'
 4329 on the PC, equipped to handle a wide range of tasks in the game. You
 4330 will also be given a summary of the history that happened before the
 4331 last screenshot. You should assist in summarizing the events for
 4332 future decision-making and also propose a new subtask, which is the
 4333 most suitable subtask for the current situation, given the target
 4334 task.

4335 Here is some helpful information to help you do the summarization and
 4336 propose the subtask.

4337 Current task:
 4338 <\$task_description\$>

4339 Previous proposed subtask for the task:
 4340 <\$subtask_description\$>

4341 Previous reasoning for proposing the subtask:
 4342 <\$subtask_reasoning\$>

4343 <\$image_introduction\$>

4344 Current budget:
 4345 <\$budget\$>

4346 Current population:
 4347 <\$population\$>

4348 Last executed action:
 4349 <\$actions\$>

4350 Self-reflection for the last executed action:
 4351 <\$self_reflection_reasoning\$>

4352 Error message for the last action:
 4353 <\$error_message\$>

4354 The following is the summary of history that happened before the last
 4355 screenshot:
 4356 <\$previous_summarization\$>

4357 The task can be decomposed into the following subtasks:

- 4358 1. Start from the Highway entry: Build a road from the highway entry in
 4359 grid (4, 2) vertically northwards towards grid (3,1).
- 4360 2. Extend Horizontally to the Left (1,1): From the endpoint in grid (1,1)
 4361 , construct a road horizontally to the left, spanning across grids
 4362 (3,1) and (2,1), and ending at the center of grid (1,1).
- 4363 3. Build a Road Down to the bottom of Grid (2,2): Start from grid (1,1)
 4364 and construct the road to the top of grid (2, 3).
- 4365 4. Extend Eastward to Grid (3,3): From the bottom of grid (2,2), build a
 4366 road eastward to reach the center of grid (3,3).
- 4367 5. Connect the road to the Highway Exit: Extend the end of the road from
 4368 grid (3,3) to the exit of the highway, completing the road loop.
- 4369 6. Install a Water Pumping Station near the River at the top-left corner
 4370 of grid (2,3): Place the water pumping station near the river in grid
 4371 (2,3) to ensure an adequate water supply.

- 4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4420
4421
4422
4423
4424
4425
4426
4427
7. Position a Water Drain Pipe near the River at the top-left corner of grid (2,3): Install a water drain pipe slightly downstream from the pumping station but within the same grid to prevent water contamination.
 8. Lay Water Pipes: Connect the water pumping station to the water drain pipe using water pipes. Additionally, ensure all roads built are covered with water pipes to provide water access across the entire area.
 9. Erect Wind Turbines for Power: Construct several wind turbines near the water pumping station and along the roads to provide sustainable electricity to the area.
 10. Designate Residential Zones: Allocate spaces adjacent to the roads for residential zones to foster community living.
 11. Establish Industrial Zones: Set aside areas near the roads for industrial purposes, ideally in parts of the grid further from residential zones to manage noise and pollution.
 12. Create Commercial Zones: Develop commercial zones near the roads to provide services and retail options for the residents and workers in the area.
 13. Make sure all the zones near roads are built with Residential Zones, Industrial Zones or Industrial Zones.
 14. Build more roads and zones and ensure water and electricity supply.
- History_summary: Summarize what happened in the past experience, especially the last step according to the decision-making reasoning and self-reflection reasoning for the last executed action. The summarization needs to be precise, concrete and highly related to the task and follow the rules below.
1. Summarize the tasks from the history and the current task. What is the current progress of the task?
 2. Which subtask has been completed? Which subtasks are not?
- Subtask_reasoning: According to the task decomposition, analyze the current progress step by step and then decide whether the previous subtask is finished and whether it is necessary to propose a new subtask. The subtask should be straightforward, contribute to the target task and be most suitable for the current situation, which should be completed within a few actions. You should respond to me with:
1. What is the previous subtask? Which step it is for in the task decomposition?
 2. According to the reasoning of self-reflection, is the previous subtask completed? Note that the success of the action does not mean the success of the subtask. You should strictly follow the reasoning of whether the subtask is completed in the self-reflection. If yes, you should move to the next step and propose it as the new subtask. If not, you should continue the previous subtask without changing anything. Please do not make any assumptions if they are not mentioned in the above information. You should assume that you are doing the task from scratch. Please strictly follow the description and requirements in the current task.
 3. The proposed subtask needs to be precise and concrete within one sentence. It should not be related to any skills.
 4. To enable water supply, you should first build a water pumping station and then build a water drain pipe near the river, and finally use water pipes to connect them with the roads. And ensure the water pipes cover all the roads.
 5. The water pumping station and water drain pipe also need electricity to work. So you also need to provide electricity for them.
 6. If you want to build roads for the village at the beginning, make sure to mention that the road needs to be as long as possible and use several roads to form a large square for the village.

4428 Subtask: According to the subtask reasoning, determine and output the
 4429 most suitable subtask for the current situation. You MUST output the
 4430 subtask in the output.
 4431
 4432 You should only respond in the format described below, and you should not
 4433 output comments or other information.
 4434 History_summary:
 4435 The summary is ...
 4436 Subtask_reasoning:
 4437 1. ...
 4438 2. ...
 4439 3. ...
 4440 Subtask:
 4441 The current subtask is ...

4441 **Prompt 9: Skylines: Action Planning prompt.**

4442
 4443 You are a helpful AI assistant integrated with 'Cities: Skylines' on the
 4444 PC, equipped to handle various tasks in the game. Your advanced
 4445 capabilities enable you to process and interpret gameplay screenshots
 4446 and other relevant information. By analyzing these inputs, you gain
 4447 a comprehensive understanding of the current context and situation
 4448 within the game. Utilizing this insight, you are tasked with
 4449 identifying the most suitable in-game action to take next, given the
 4450 current task. You control the game character and can execute actions
 4451 from the available action set. Upon evaluating the provided
 4452 information, your role is to articulate the precise action you would
 4453 deploy, considering the game's present circumstances, and specify any
 4454 necessary parameters for implementing that action.
 4455
 4456 Here is some helpful information to help you make the decision.
 4457
 4458 Current task:
 4459 <\$subtask_description\$>
 4460
 4461 Coordinates of constructed buildings:
 4462 <\$coordinates\$>
 4463
 4464 The latest successful action that builds the building. If you want to
 4465 try_place a road, and the endpoint (x2, y2), of the latest successful
 4466 action is also try_place a road. Then you MUST use the end point of
 4467 the constructed road as the start point of your new road.
 4468 <\$last_success_try_place_action\$>
 4469
 4470 Current budget:
 4471 <\$budget\$>
 4472
 4473 Current population:
 4474 <\$population\$>
 4475
 4476 Last executed action:
 4477 <\$actions\$>
 4478
 4479 Self-reflection reasoning for the last executed action:
 4480 <\$self_reflection_reasoning\$>
 4481
 4482 Error message for the last action:
 4483 <\$error_message\$>
 4484
 4485 Construction information for the last action:
 4486 <\$construction_information\$>
 4487
 4488 Summarization of recent history:
 4489 <\$history_summary\$>
 4490

4482 Valid action set in Python format to select the next action:
 4483 <\$skill_library\$>
 4484
 4485 <\$image_introduction\$>
 4486
 4487 Based on the above information, analyze the current situation and provide
 4488 the reasoning for what you should do for the next step to complete
 4489 the task. Then, you should output the exact action you want to
 4490 execute in the game. You should respond to me with:
 4491
 4492 Reasoning: You should think step by step and provide detailed reasoning
 4493 to determine the next action executed on the current state of the
 4494 task. You need to answer the following questions step by step. You
 4495 cannot miss the last question:
 4496 1. What is the current task? What are the requirements to achieve the
 4497 goal?
 4498 2. According to the self-reflection reasoning, is the last action
 4499 executed successfully?
 4500 3. If you want to place anything, do you already open the
 4501 corresponding menu? Otherwise, you need to open the right menu first
 4502 in this step rather than doing anything else. If you have not already
 4503 opened the corresponding menu, skip answering questions 4, 5, 6, 7,
 4504 8 and 9.
 4505 4. Does the previous action "try_place" something? If there is an
 4506 error message showing that the space is already occupied or the last
 4507 action failed according to the self-reflection reasoning, you should
 4508 use the same action with different parameters as the position of it
 4509 to try again. The difference needs to be significant enough with at
 4510 least 100 pixels of change for the position of the input points. If
 4511 there is no error message, you should only output confirm_placement()
 4512 or cancel_placement() to approve or cancel the placement. You should
 4513 not call anything else.
 4514 5. Does the previous action open any menu? Then you should "try_place
 4515 " something according to the task description instead of using "
 4516 confirm_placement".
 4517 6. If you want to place a building, which grid do you plan to place
 4518 the building in? What is the exact pixel position of it?
 4519 7. If you want to place a road, which grids do you plan to make it
 4520 cross? Which grids are the start point and end point in, respectively
 4521 ? What are the exact pixel positions of them? You MUST use one of the
 4522 endpoints of the constructed road shown in the coordinates
 4523 information as the start point of the new road. If you want to
 4524 try_place a road, and the endpoint (x2, y2), of the latest successful
 4525 action is also try_place a road. Then you MUST use the end point of
 4526 the constructed road as the start point of your new road.
 4527 8. If you want to place a zone, which grids do you plan to make it
 4528 cover? You should only use the vertices coordinates of the
 4529 corresponding grids as the parameter for the action. Zones cannot
 4530 cover each other.
 4531 9. If you want to place a Water Pipe, the start point should be the
 4532 position of Water Pumping Station, Water Drain Pipe, the start point
 4533 of a built Water Pipe or the end point of a built Water Pipe.
 4534 10. This is the most critical question. Based on the action rules and
 4535 self-reflection, what should be the most suitable action in the
 valid action set for the next step? You should analyze the effects of
 the action step by step. You should not repeat the previous action
 again. Do not try to verify whether the previous action succeeded.
 11. Do all the selected actions exist in the valid action set? If no,
 regenerate the action and give the reasons.
 12. If you are placing a road, is the road more than 300 pixels long?
 Otherwise, regenerate the action and give reasons.

Actions: The requirements that the generated action needs to follow. The
 best action, or short sequence of actions without gaps, to execute
 next to progress in achieving the goal. Pay attention to the names of

4536 the available skills and to the previous skills already executed, if
 4537 any. You should also pay more attention to the following action
 4538 rules:

- 4539 1. You should output actions in Python code format and specify any
 4540 necessary parameters to execute that action. If the function has
 4541 parameters, you should also include their names and decide their
 4542 values, like "move_right(duration=1)". If it does not have a
 4543 parameter, just output the action, like "open_map()".
- 4544 2. Given the current situation and task, you should only choose the
 4545 most suitable action from the valid action set. You cannot use
 4546 actions that are not in the valid action set to control the character
 4547 .
- 4548 3. You MUST NOT output more than one skill in the actions.
- 4549 4. If you want to build a village, you should follow these rules:
 4550 4.1 Build roads correctly.
 4551 - If you have not opened the road tool, you should open the menu.
 4552 If you have already opened the menu, you should not open it again.
 4553 - Newly built roads must be connected to the existing roads.
 4554 - Determine in which grid the starting point of the newly built
 4555 road is located, and identify the pixel position of the starting
 4556 point.
 4557 - Build the road in the correct direction.
- 4558 5. You MUST NOT repeat the previous action with the same parameters
 4559 again if you think the previous action fails.
- 4560 6. Your action should strictly follow the analysis in the reasoning.
 4561 Do not output any additional action not mentioned in the reasoning.
- 4562 7. Please do not directly connect the entrance of the highway with
 4563 the exit of the highway at the beginning. To make the village as
 4564 large as possible. You should build roads in the wild and connect
 4565 them with each other.
- 4566 8. If you are placing a road, the road needs to be at least 300
 4567 pixels long.

4568 You should only respond in the format described below, and you should not
 4569 output comments or other information.

4570 Reasoning:

4571 1. ...
 4572 2. ...
 4573 3. ...

4574 ...

4575 Actions:

```
4576 ```python
4577     action(args1=x,args2=y)
4578 ```
```

4579 K.3 PROMPTS FOR STARDEW VALLEY

4580 Prompt 10: Stardew: Information Gathering Cultivation prompt.

4581 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
 4582 the PC, equipped to handle a wide range of tasks in the game. Your
 4583 advanced capabilities enable you to process and interpret gameplay
 4584 screenshots and other relevant information.

4585 <\$image_introduction\$>

4586 Current task:

4587 <\$task_description\$>

4588 Description: Please analyze and describe the screenshot image in a grid-
 4589 by-grid format and then provide an overall image description. Pay
 attention to anything related to the task. The image is divided into
 a 3x5 grid, each cell having its own coordinates. For each grid cell,
 describe the contents in detail, focusing on any critical icons, or

4590 objects present in that particular segment. If there are specific
 4591 features such as characters or text, mention these as well. After
 4592 completing the description for one cell, proceed to the next, for
 4593 example, 'In grid (1,1), [description]. In grid (1,2), [description
 4594].' and so on until the entire image is covered.

4595 Date_time: The date and time information in the game are shown on the
 4596 upper-right of the screenshot, in grid (1, 5). An example of the date
 4597 and time information is "Wed 10, 5:10 pm".

4598 Energy: The current energy remains for the character doing actions. The
 4599 energy bar is shown on the bottom-right of the screenshot, in grid
 4600 (3, 5). The full energy is 270. An example of the energy information
 4601 is "150/270".

4602 Weather: The current weather information in the game, the weather is one
 4603 from "Sunny", "Rainy", "Windy", "Snowy", "Stormy", "Festival", "
 4604 Wedding", and "null". If none of them applies, only output "null".

4605 Dialog: If there are some dialogs shown in the screenshot, extract the
 4606 text of the conversation, like "Shopkeeper: What do you want to buy
 4607 ?", otherwise, only output "null".

4608 Other: Other information that does not belong to the above categories. If
 4609 none of them applies, only output "null".

4610 You should only respond in the format described below and not output
 4611 comments or other information.

4612 Description:
 4613 In grid (1,1), ...
 4614 In grid (1,2), ...
 4615 ...
 4616 In grid (3,5), ...
 4617 Overall, the image shows...

4618 Date_time:
 4619 Date and time information
 4620 Energy:
 4621 The number of energy remains showing in the energy bar
 4622 Weather:
 4623 Weather information
 4624 Dialog:
 4625 Dialog text
 4626 Other:
 4627 Other information is ...

4628 **Prompt 11: Stardew: Self-Reflection Cultivation prompt.**

4629 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
 4630 the PC, equipped to handle a wide range of tasks in the game. Your
 4631 advanced capabilities enable you to process and interpret gameplay
 4632 screenshots and other relevant information. Your task is to examine
 4633 these inputs, interpret the in-game context, and determine whether
 4634 the executed action takes effect.

4635 Target task:
 4636 <\$task_description\$>

4637 Current subtask for completing the target task:
 4638 <\$subtask_description\$>

4639 The reasoning for proposing the current subtask:
 4640 <\$subtask_reasoning\$>

4641 Last executed action for completing the subtask:
 4642 <\$previous_action\$>

4644
4645 Reasoning for the last action:
4646 <\$previous_reasoning\$>
4647
4648 Current date and time:
4649 <\$date_time\$>
4650
4651 Previous toolbar information:
4652 <\$previous_toolbar_information\$>
4653
4654 Current toolbar information:
4655 <\$toolbar_information\$>
4656
4657 Summarization of recent history:
4658 <\$history_summary\$>
4659
4660 <\$image_introduction\$>
4661 Reasoning: You need to answer the following questions step by step to get
4662 some reasoning based on the last action and sequential frames of the
4663 character during the execution of the last action.
4664 1. What is the executed action? Please answer this question not based on
4665 the sequential frames.
4666 2. Was the executed action successful? Give reasons. You should refer to
4667 the following rules:
4668 - If the action involves moving forward, it is considered unsuccessful
4669 only when the character's position remains unchanged across
4670 sequential frames, regardless of background elements and other people
4671 .
4672 - If you are not 100% sure that the action fails, regard it as success.
4673 3. If the last action is not executed successfully, what is the most
4674 probable cause? You should give only one cause and refer to the
4675 following rules:
4676 - The reasoning for the last action could be wrong.
4677 - If it is an interaction action, the most probable cause was that the
4678 action was unavailable at the current place, then you should move to
4679 a new place.
4680 - If it is a movement action, the most probable cause was that you were
4681 blocked by seen or unseen obstacles.
4682 - If there is an error report, analyze the cause based on the report.
4683 4. Is the subtask completed? Give your reasons. If you want to make any
4684 confirmation, regard it as a success.
4685 5. Is the target task completed? Give your reasons.
4686 6. Do you think the subtask is reasonable? Give your reasons.
4687
4688 You should only respond in the format described below.
4689 Reasoning:
4690 1. ...
4691 2. ...
4692 3. ...
4693 ...
4694 ...
4695 ...

Prompt 12: Stardew: Task Inference Cultivation prompt.

4689 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
4690 the PC, equipped to handle a wide range of tasks in the game. You
4691 will also be given a summary of the history that happened before the
4692 last screenshot. You should assist in summarizing the events for
4693 future decision-making and also propose a new subtask, which is the
4694 most suitable subtask for the current situation, given the target
4695 task.
4696 Here is some helpful information to help you do the summarization and
4697 propose the subtask.

4698 Current task:
4699 <\$task_description\$>
4700

4701 Previous proposed subtask for the task:
4702 <\$subtask_description\$>
4703

4704 Previous reasoning for proposing the subtask:
4705 <\$subtask_reasoning\$>
4706

4707 <\$image_introduction\$>
4708

4709 Current toolbar information:
4710 <\$toolbar_information\$>
4711

4712 Last executed action:
4713 <\$previous_action\$>
4714

4715 Decision-making reasoning for the last executed action:
4716 <\$previous_reasoning\$>
4717

4718 Self-reflection for the last executed action:
4719 <\$self_reflection_reasoning\$>
4720

4721 The following is the summary of history that happened before the last
4722 screenshot:
4723 <\$previous_summarization\$>
4724

4725 History_summary: Summarize what happened in the past experience,
4726 especially the last step according to the decision-making reasoning
4727 and self-reflection reasoning for the last executed action. The
4728 summarization needs to be precise, concrete and highly related to the
4729 task and follow the rules below.
4730 1. Summarize the tasks from the history and the current task. What is the
4731 current progress of the task? For example, to harvest a seed, you
4732 need to water the seed for 4 days. And you have already planted the
4733 seed and watered it for two days.
4734 2. Record the successful actions and organize them into events day by day
4735 .
4736 3. Do not forget the information and key events in the previous days.
4737 4. If you are watering a seed. Record how many times you have watered and
4738 calculate how many days you have to water before you can harvest
4739 according to the toolbar information provided above.

4740 Here is an example to follow:
4741 On Thu.4, I dig the dirt with the toe and then plant the parsnip seed and
4742 water the seed. The seed has been watered once. It still needs to be
4743 watered another threetimes to harvest. On Fri.5, I watered the seed
4744 again. The seed has been watered twice. It still needs to be watered
4745 twice to harvest. Today, Sat.6, I just need to get out of home and
4746 watered the seed again.
4747

4748 Subtask_reasoning: Decide whether the previous subtask is finished and
4749 whether it is necessary to propose a new subtask. The subtask should
4750 be straightforward, contribute to the target task and be most
4751 suitable for the current situation, which should be completed within
a few actions. You should respond to me with:

1. How to finish the target task? You should analyze it step by step.
2. What is the current progress of the target task according to the analysis in step 1? Please do not make any assumptions if they are not mentioned in the above information. You should assume that you are doing the task from scratch.
3. What is the previous subtask? Does the previous subtask finish? Or is it improper for the current situation? Then select a new one, otherwise you should reuse the last subtask.
4. If you want to propose a new subtask, give reasons why it is more feasible for the current situation.

4752 5. The proposed subtask needs to be precise and concrete within one
 4753 sentence. It should not be related to any skills.
 4754 6. The seed only needs to be watered once.
 4755 7. Do not mention any grid information in the subtask description.
 4756 8. Do not check the growth status of the crop.
 4757 9. The seeds only need to be watered ONCE every day. If you have already
 4758 watered the seed today, you should return home and go to sleep,
 4759 waiting for the next day.

4760 You should only respond in the format described below, and you should not
 4761 output comments or other information.

4762 History_summary:
 4763 The summary is...
 4764 Subtask_reasoning:
 4765 1. ...
 4766 2. ...
 4767 ...
 4768 Subtask:
 4769 The current subtask is

4769 **Prompt 13: Stardew: Action Planning Cultivation prompt.**

4770

4771 You are a helpful AI assistant integrated with 'Stardew Valley' on the PC
 4772 , equipped to handle various tasks in the game. Your advanced
 4773 capabilities enable you to process and interpret gameplay screenshots
 4774 and other relevant information. By analyzing these inputs, you gain
 4775 a comprehensive understanding of the current context and situation
 4776 within the game. Utilizing this insight, you are tasked with
 4777 identifying the most suitable in-game action to take next, given the
 4778 current task. You control the game character and can execute actions
 4779 from the available action set. Upon evaluating the provided
 4780 information, your role is to articulate the precise action you would
 4781 deploy, considering the game's present circumstances, and specify any
 4782 necessary parameters for implementing that action.

4783 Here is some helpful information to help you make the decision.

4784 Current subtask:
 4785 <\$subtask_description\$>

4786 Current date and time:
 4787 <\$date_time\$>

4788 Toolbar information:
 4789 <\$toolbar_information\$>

4790 Last executed action:
 4791 <\$previous_action\$>

4792 Reasoning for the last action:
 4793 <\$previous_reasoning\$>

4794 Self-reflection for the last executed action:
 4795 <\$previous_self_reflection_reasoning\$>

4796 Summarization of recent history:
 4797 <\$history_summary\$>

4800 Valid action set in Python format to select the next action:
 4801 <\$skill_library\$>

4802 <\$image_introduction\$>

4803 Based on the above information, analyze the current situation and provide
 4804 the reasoning for what you should do for the next step to complete

4806 the task. Then, you should output the exact action you want to
4807 execute in the game. You should respond to me with:
4808

Reasoning: You should think step by step and provide detailed reasoning
4809 to determine the next action executed on the current state of the
4810 task. You need to answer the following questions step by step. You
4811 cannot miss the last question:
4812 1. Analyze the information in the toolbar. Does it contain all the
4813 necessary items for completing the task?
4814 2. What is the current selected tool? Do you want to use a tool, such
4815 as axe, hoe, watering can, pickaxe and scythe? And is the character'
4816 s current position a suitable place to use such a tool? Then you
4817 should use use_tool() instead of do_action().
4818 3. Does the character already reach the target place?
4819 4. What was the previous action? If the previous action was a
4820 movement, were you blocked?
4821 5. If your task is to harvest the plant, did you water the seed? The
4822 seeds only need to be watered ONCE every day. If you have already
4823 watered the seed today, you should return home and go to sleep,
4824 waiting for the next day.
4825 6. This is the most critical question. Based on the action rules and
4826 self-reflection, what should be the most suitable action in the valid
4827 action set for the next step? You should analyze the effects of the
4828 action step by step. You should not repeat the previous action again
4829 except for the movement action. Do not try to verify whether the
4830 previous action succeeded.
4831 7. Is the selected action the same as the last executed action? If
4832 yes, regenerate the action and give the reasons.
4833 8. Do all the selected actions exist in the valid action set? If no,
4834 regenerate the action and give the reasons.
4835 9. Analyze whether the selected action meets the requirements of the
4836 Actions below one by one. Does the generated action meet all the
4837 requirements? If not, regenerate the action and give the reasons.
4838

Actions: The requirements that the generated action needs to follow. The
4839 best action, or short sequence of actions without gaps, to execute
4840 next to progress in achieving the goal. Pay attention to the names of
4841 the available skills and to the previous skills already executed, if
4842 any. You should also pay more attention to the following action
4843 rules:
4844 1. You should output actions in Python code format and specify any
4845 necessary parameters to execute that action. If the function has
4846 parameters, you should also include their names and decide their
4847 values, like "move_right(duration=1)". If it does not have a
4848 parameter, just output the action, like "open_map()".
4849 2. You can only output at most two actions in the output.
4850 3. In the screenshots, the blue band represents the left side and the
4851 yellow band represents the right side. Please ignore character's
4852 facing direction and output the action in an absolute direction like
4853 right and left.
4854 4. If you want to interact with the objects in the toolbar, you need
4855 to make sure that the target object is already selected. You need to
4856 use select_tool() to select them before executing use_tool() or
4857 do_action().
4858 5. If you want to plant a seed or harvest a mature crop, please use
4859 do_action() instead of use_tool(). If you want to use tools, like axe
4860 , hoe, watering can, pickaxe and scythe, please use use_tool().
4861 6. If upon self-reflection you think the last action was unavailable
4862 at the current place, you MUST move to another place. Please do not
4863 try to execute the same action again.
4864 7. If you want to get out of the house, just use the skill
4865 get_out_of_house(). You MUST NOT output any movement action behind
4866 this skill. And if the last executed action already contains this
4867 skill, do not execute this skill for the current step again.

4860 8. If upon self-reflection you think you were blocked, you MUST
 4861 change the direction of moving, so that you can pass obstacles.
 4862 9. You MUST NOT repeat the previous action again if you think the
 4863 previous action fails.
 4864 10. Your action should strictly follow the analysis in the reasoning.
 4865 Do not output any additional action not mentioned in the reasoning.

4866 You should only respond in the format described below, and you should not
 4867 output comments or other information.

4868 Reasoning:
 4869 1. ...
 4870 2. ...
 4871 3. ...

4872 Actions:
 4873 ```python
 4874 action(args1=x,args2=y)
 4875 ```

4876 Prompt 14: Stardew: Information Gathering Farm Cleanup prompt.

4877 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
 4878 the PC, equipped to handle a wide range of tasks in the game. Your
 4879 advanced capabilities enable you to process and interpret gameplay
 4880 screenshots and other relevant information.

4881 <\$image_introduction\$>
 4882

4883 Current task:
 4884 <\$task_description\$>

4885 Description: Please analyze and describe the screenshot image in a grid-
 4886 by-grid format and then provide an overall image description. Pay
 4887 attention to anything related to the task. The image is divided into
 4888 a 3x5 grid, each cell having its own coordinates. For each grid cell,
 4889 describe the contents in detail, focusing on any critical icons, or
 4890 objects present in that particular segment. If there are specific
 4891 features such as characters or text, mention these as well. After
 4892 completing the description for one cell, proceed to the next, for
 4893 example, 'In grid (1,1), [description]. In grid (1,2), [description
 4894].' and so on until the entire image is covered.

4895 Date_time: The date and time information in the game are shown on the
 4896 upper-right of the screenshot, in grid (1, 5). An example of the date
 4897 and time information is "Wed 10, 5:10 pm".

4898 Energy: The current energy remains for the character doing actions. The
 4899 energy bar is shown on the bottom-right of the screenshot, in grid
 4900 (3, 5). The full energy is 270. An example of the energy information
 4901 is "150/270".

4902 Weather: The current weather information in the game, the weather is one
 4903 from "Sunny", "Rainy", "Windy", "Snowy", "Stormy", "Festival", "
 4904 Wedding", and "null". If none of them applies, only output "null".

4905 Dialog: If there are some dialogs shown in the screenshot, extract the
 4906 text of the conversation, like "Shopkeeper: What do you want to buy
 4907 ?", otherwise, only output "null".

4908 Other: Other information that does not belong to the above categories. If
 4909 none of them applies, only output "null".

4910

4911 You should only respond in the format described below and not output
 4912 comments or other information.

4913 Description:
 In grid (1,1), ...

4914 In grid (1,2), ...
 4915 ...
 4916 In grid (3,5), ...
 4917 Overall, the image shows...
 4918 Date_time:
 4919 Date and time information
 4920 Energy:
 4921 The number of energy remains showing in the energy bar
 4922 Weather:
 4923 Weather information
 4924 Dialog:
 4925 Dialog text
 4926 Other:
 4927 Other information is ...

4927 **Prompt 15: Stardew: Self-Reflection Farm Cleanup prompt.**

4928 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
 4929 the PC, equipped to handle a wide range of tasks in the game. Your
 4930 advanced capabilities enable you to process and interpret gameplay
 4931 screenshots and other relevant information. Your task is to examine
 4932 these inputs, interpret the in-game context, and determine whether
 4933 the executed action takes effect.
 4934 Target task:
 4935 <\$task_description\$>
 4936
 4937 Current subtask for completing the target task:
 4938 <\$subtask_description\$>
 4939
 4940 The reasoning for proposing the current subtask:
 4941 <\$subtask_reasoning\$>
 4942
 4943 Last executed action for completing the subtask:
 4944 <\$previous_action\$>
 4945
 4946 Reasoning for the last action:
 4947 <\$previous_reasoning\$>
 4948
 4949 Current date and time:
 4950 <\$date_time\$>
 4951
 4952 Previous toolbar information:
 4953 <\$previous_toolbar_information\$>
 4954
 4955 Current toolbar information:
 4956 <\$toolbar_information\$>
 4957
 4958 Summarization of recent history:
 4959 <\$history_summary\$>
 4960
 4961 <\$image_introduction\$>
 4962 Reasoning: You need to answer the following questions step by step to get
 4963 some reasoning based on the last action and sequential frames of the
 4964 character during the execution of the last action.
 4965 1. What is the executed action? Please answer this question not based on
 4966 the sequential frames.
 4967 2. Was the executed action successful? Give reasons. You should refer to
 the following rules:
 - If the action involves moving forward, it is considered unsuccessful
 only when the character's position remains unchanged across
 sequential frames, regardless of background elements and other people
 .
 - If you are not 100% sure that the action fails, regard it as success.

4968 3. If the last action is not executed successfully, what is the most
4969 probable cause? You should give only one cause and refer to the
4970 following rules:
4971 - The reasoning for the last action could be wrong.
4972 - If it is an interaction action, the most probable cause was that the
4973 action was unavailable at the current place, then you should move to
4974 a new place.
4975 - If it is a movement action, the most probable cause was that you were
4976 blocked by seen or unseen obstacles.
4977 - If there is an error report, analyze the cause based on the report.
4978 4. Is the subtask completed? Give your reasons. If you want to make any
4979 confirmation, regard it as a success.
4980 5. Is the target task completed? Give your reasons.
4981 6. Do you think the subtask is reasonable? Give your reasons.
4982
4983 You should only respond in the format as described below.
4984 Reasoning:
4985 1. ...
4986 2. ...
4987 3. ...
4988 ...

Prompt 16: Stardew: Task Inference Farm Cleanup prompt.

4988 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
4989 the PC, equipped to handle a wide range of tasks in the game. You
4990 will also be given a summary of the history that happened before the
4991 last screenshot. You should assist in summarizing the events for
4992 future decision-making and also propose a new subtask, which is the
4993 most suitable subtask for the current situation, given the target
4994 task.
4995
4996 Here is some helpful information to help you do the summarization and
4997 propose the subtask.
4998
4999 Current task:
5000 <\$task_description\$>
5001
5002 Previous proposed subtask for the task:
5003 <\$subtask_description\$>
5004
5005 Previous reasoning for proposing the subtask:
5006 <\$subtask_reasoning\$>
5007
5008 <\$image_introduction\$>
5009
5010 Current toolbar information:
5011 <\$toolbar_information\$>
5012
5013 Last executed action:
5014 <\$previous_action\$>
5015
5016 Decision-making reasoning for the last executed action:
5017 <\$previous_reasoning\$>
5018
5019 Self-reflection for the last executed action:
5020 <\$self_reflection_reasoning\$>
5021
5022 The following is the summary of history that happened before the last
5023 screenshot:
5024 <\$previous_summarization\$>
5025
5026 History_summary: Summarize what happened in the past experience,
5027 especially the last step according to the decision-making reasoning
5028 and self-reflection reasoning for the last executed action. The

5022 summarization needs to be precise, concrete and highly related to the
 5023 task and follow the rules below.

- 5024 1. Summarize the tasks from the history and the current task. What is the
 5025 current progress of the task? For example, to harvest a seed, you
 5026 need to water the seed for 4 days. And you have already planted the
 5027 seed and watered it for two days.
- 5028 2. Record the successful actions and organize them into events day by day
 5029 .
- 5029 3. Do not forget the information and key events in the previous days.
- 5030 4. If you are watering a seed. Record how many times you have watered and
 5031 calculate how many days you have to water before you can harvest
 5032 according to the toolbar information provided above.

5033 Here is an example to follow:
 5034 On Thu.4, I dig the dirt with the toe and then plant the parsnip seed and
 5035 water the seed. The seed has been watered once. It still needs to be
 5036 watered another three times to harvest. On Fri.5, I watered the seed
 5037 again. The seed has been watered twice. It still needs to be watered
 5038 twice to harvest. Today, Sat.6, I just need to get out of home and
 5039 watered the seed again.

5039 Subtask_reasoning: Decide whether the previous subtask is finished and
 5040 whether it is necessary to propose a new subtask. The subtask should
 5041 be straightforward, contribute to the target task and be most
 5042 suitable for the current situation, which should be completed within
 5043 a few actions. You should respond to me with:

- 5044 1. How to finish the target task? You should analyze it step by step.
- 5045 2. What is the current progress of the target task according to the
 5046 analysis in step 1? Please do not make any assumptions if they are
 5047 not mentioned in the above information. You should assume that you
 5048 are doing the task from scratch.
- 5048 3. What is the previous subtask? Does the previous subtask finish? Or is
 5049 it improper for the current situation? Then select a new one,
 5050 otherwise you should reuse the last subtask.
- 5050 4. If you want to propose a new subtask, give reasons why it is more
 5051 feasible for the current situation.
- 5052 5. The proposed subtask needs to be precise and concrete within one
 5053 sentence. It should not be related to any skills.
- 5054 6. The seed only needs to be watered once.
- 5054 7. Do not mention any grid information in the subtask description.
- 5055 8. Do not check the growth status of the crop.
- 5056 9. The seeds only need to be watered ONCE every day. If you have already
 5057 watered the seed today, you should return home and go to sleep,
 5058 waiting for the next day.

5059 You should only respond in the format described below, and you should not
 5060 output comments or other information.

5061 History_summary:
 5062 The summary is...
 5063 Subtask_reasoning:
 5064 1. ...
 5064 2. ...
 5065 ...
 5066 Subtask:
 5067 The current subtask is

5068 **Prompt 17: Stardew: Action Planning Farm Cleanup prompt.**

5070 You are a helpful AI assistant integrated with 'Stardew Valley' on the PC
 5071 , equipped to handle various tasks in the game. Your advanced
 5072 capabilities enable you to process and interpret gameplay screenshots
 5073 and other relevant information. By analyzing these inputs, you gain
 5074 a comprehensive understanding of the current context and situation
 5075 within the game. Utilizing this insight, you are tasked with
 identifying the most suitable in-game action to take next, given the
 current task. You control the game character and can execute actions

5076 from the available action set. Upon evaluating the provided
5077 information, your role is to articulate the precise action you would
5078 deploy, considering the game's present circumstances, and specify any
5079 necessary parameters for implementing that action.

5080 Here is some helpful information to help you make the decision.
5081

5082 Current subtask:
5083 <\$subtask_description\$>
5084

5085 Current date and time:
5086 <\$date_time\$>
5087

5087 Toolbar information:
5088 <\$toolbar_information\$>
5089

5090 Last executed action:
5091 <\$previous_action\$>
5092

5092 Reasoning for the last action:
5093 <\$previous_reasoning\$>
5094

5095 Self-reflection for the last executed action:
5096 <\$previous_self_reflection_reasoning\$>
5097

5097 Summarization of recent history:
5098 <\$history_summary\$>
5099

5100 Valid action set in Python format to select the next action:
5101 <\$skill_library\$>
5102
5103 <\$image_introduction\$>
5104

5104 Based on the above information, analyze the current situation and provide
5105 the reasoning for what you should do for the next step to complete
5106 the task. Then, you should output the exact action you want to
5107 execute in the game. You should respond to me with:

5108 Reasoning: You should think step by step and provide detailed reasoning
5109 to determine the next action executed on the current state of the
5110 task. You need to answer the following questions step by step. You
5111 MUST NOT miss question 3 and question 11:
5112 1. Analyze the information in the tool bar. Does it contain all the
5113 necessary items for completing the task?
5114 2. Where is the character in the screenshot of the current step?
5115 Where is the house in the screenshot of the current step? The blue
5116 band represents the left side and the yellow band represents the
5117 right side. Where is the character compared with the house? (Is he at
5118 the left edge or right edge of the house?)
5119 3. If your task is to clear obstacles, you MUST NOT miss any question
5120 in this step:
5121 - The blue band represents the left side and the yellow band
5122 represents the right side. Where is the character according to the
5123 house? (Is he at the left edge or right edge of the house?)
5124 - Which grids do the house span in the screenshot? (You MUST answer
5125 one or two grid position. The house does not span over two grids.)
5126 Then, what are the two grids below and near the house? (e.g. If the
5127 house spans from grid (1,3) to (1,4), the CLEARING AREA of character
5128 should be grid (2,3) and (2,4). If the house spans grid (1,3), the
5129 CLEARING AREA of character should be grid (2,2) and (2,3). You MUST
remember this CLEARING AREA precisely IN THIS ROUND.) You should
focus on obstacles in them. You MUST NOT move the character out of
these two obstacle grids.
- In order to clear all obstacles below the house and make the
place suitable for cultivating, you should not target for a specific

5130 obstacle. Instead, you should try your best to move the character to
5131 pass every patch in the CLEARING AREA. You should clear every
5132 obstacle that blocks the character in this process.

- 5133 - Every time after you move the character down (or up when being
5134 too far from the house), you should move the character right or left
5135 (based on the character's position in the CLEARING AREA compared with
5136 the house) to fully explore the CLEARING AREA of the two grids
5137 determined above. You should clear all obstacles the character meets
5138 in this process.
- 5139 - Is the current row fully explored by the character? If so, your
5140 movement should be moving down. If there is an obstacle beneath the
5141 character, you should clear it first before moving the character down
5142 .
- 5143 - You should not move too far from the house. You should not move
5144 the character down but should move him up instead if the house is not
5145 in the current screenshot.
- 5146 - What was the previous action? If the previous action contained
5147 use_tool(), you MUST NOT start with the same use_tool() action in
5148 this round. (You can still use use_tool() by following a movement or
5149 select_tool().)
- 5150 - If the previous action was a movement, is the position of
5151 character changed? If not, it is the most trustworthy evidence that
5152 there is an obstacle in front of the character that can interact with
5153 .
- 5154 - If the character is blocked by an obstacle in front of him or if
5155 you think there is an obstacle in front of the character, what type
5156 of obstacle is it? (Usually, weed and grass are green, stone is grey
5157 and branch is brown) What is the suitable tool for clearing it and is
5158 the tool correctly selected?
- 5159 4. What is the current selected tool? Do you want to use a tool, such
5160 as axe, hoe, watering can, pickaxe and scythe? And is the character's
5161 current position a suitable place to use such a tool? Then you
5162 should use use_tool() instead of do_action().
- 5163 5. Does the character already reach the target place?
- 5164 6. What was the previous action? If the previous action was a
5165 movement, were you blocked?
- 5166 7. If your task is to harvest the plant, did you water the seed? The
5167 seeds only need to be watered ONCE every day. If you have already
5168 watered the seed today, you should return home and go to sleep,
5169 waiting for the next day.
- 5170 8. This is the most critical question. Based on the action rules and
5171 self-reflection, what should be the most suitable action in the valid
5172 action set for the next step? You should analyze the effects of the
5173 action step by step. You should not repeat the previous action again
5174 except for the movement action. Do not try to verify whether the
5175 previous action succeeded.
- 5176 9. Is the selected action the same as the last executed action? If
5177 yes, regenerate the action and give the reasons.
- 5178 10. Do all the selected actions exist in the valid action set? If no,
5179 regenerate the action and give the reasons.
- 5180 11. Analyze whether the selected action meets the requirements of the
5181 Actions below one by one. Does the generated action meet all the
5182 requirements? If not, regenerate the action and give the reasons.

5183 Actions: The requirements that the generated action needs to follow. The
5184 best action, or short sequence of actions without gaps, to execute
5185 next to progress in achieving the goal. Pay attention to the names of
5186 the available skills and to the previous skills already executed, if
5187 any. You should also pay more attention to the following action
5188 rules:

- 5189 1. You should output actions in Python code format and specify any
5190 necessary parameters to execute that action. If the function has
5191 parameters, you should also include their names and decide their
5192 values, like "move_right(duration=1)". If it does not have a
5193 parameter, just output the action, like "open_map()".

5184 2. You can only output at most two actions in the output.
5185 3. In the screenshots, the blue band represents the left side and the
5186 yellow band represents the right side. Please ignore character's
5187 facing direction and output the action in an absolute direction like
5188 right and left.
5189 4. If you want to interact with the objects in the toolbar, you need
5190 to make sure that the target object is already selected. You need to
5191 use `select_tool()` to select them before executing `use_tool()` or
5192 `do_action()`.
5193 5. If you want to plant a seed or harvest a mature crop, please use
5194 `do_action()` instead of `use_tool()`. If you want to use tool, like axe,
5195 hoe, watering can, pickaxe and scythe, please use `use_tool()`.
5196 6. If upon self-reflection you think the last action was unavailable
5197 at the current place, you MUST move to another place. Please do not
5198 try to execute the same action again.
5199 7. If you want to get out of the house, just use the skill
5200 `get_out_of_house()`. You MUST NOT output any movement action behind
5201 this skill. And if the last executed action already contains this
5202 skill, do not execute this skill for the current step again.
5203 8. If upon self-reflection you think you were blocked, you MUST
5204 change the direction of moving, so that you can pass obstacles.
5205 9. You MUST NOT repeat the previous action again if you think the
5206 previous action fails.
5207 10. Your action should strictly follow the analysis in the reasoning.
5208 Do not output any additional action not mentioned in the reasoning.
5209 11. If you want to clear obstacles, you should follow the order of
5210 thinking as follows:
5211 - You MUST NOT move the character to the house.
5212 - In order to clear all obstacles below the house and make the
5213 place suitable for cultivating, you should not target for a specific
5214 obstacle. Instead, you should try your best to move the character to
5215 pass every patch in the CLEARING AREA. You should clear every
5216 obstacle that blocks the character in this process.
5217 - Every time after you move the character down (or up when being
5218 too far from the house), you should move the character right or left
5219 (based on the character's position compared with the house) to fully
5220 explore the CLEARING AREA. You should clear all obstacles the
5221 character meets in this process.
5222 - If you think the character has fully explored the current row
5223 of the CLEARING AREA, you should move the character down. If there is
5224 an obstacle beneath the character, you should clear it first before
5225 moving the character down.
5226 - You should not move too far from the house. You should not move
5227 the character down but should move him up instead if the house is
5228 not in the current screenshot.
5229 - You can take larger steps of moving left or right by adjusting
5230 the action's parameter. You MUST use a small parameter when doing
5231 `move_down()` to make sure the character only moves one patch down.
5232 - If you think there is an obstacle in front of the character,
5233 you should determine its type. You should then select the suitable
5234 tool by `select_tool()` and clear the obstacle by `use_tool()`.
5235 - You should always `use_tool()` after `select_tool()`. Do not switch
5236 to another tool without using it.
5237 - If the previous action contained `use_tool()`, you MUST NOT start
5238 with the same `use_tool()` action in this round. (You can still use
5239 `use_tool()` by following a movement or `select_tool()`.)
5240 - If the previous action contained `use_tool()`, you should
5241 determine whether the obstacle is cleared. If you are not sure that
5242 the obstacle is cleared, you are encouraged to try different tools by
5243 `select_tool()` and `use_tool()` before moving the character to other
5244 positions.
5245 - If the previous action was a movement, you should determine
5246 whether there is an obstacle IN FRONT OF the character. If so, you
5247 should select the suitable tool by `select_tool()` and clear it by
5248 `use_tool()`.

5238 - If previous action contained use_tool(), you should move the
 5239 character to the same direction as before to test if the blocking
 5240 obstacle is cleared.
 5241 - If the blocking obstacle is not cleared, you should select a
 5242 different tool to clear it.
 5243
 5244 You should only respond in the format described below, and you should not
 5245 output comments or other information.
 5246 Reasoning:
 5247 1. ...
 5248 2. ...
 5249 3. ...
 5249 Actions:
 5250 ```python
 5251 action(args1=x,args2=y)
 5252 ```

5253
 5254 **Prompt 18: Stardew: Information Gathering Shopping prompt.**

5255 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
 5256 the PC, equipped to handle a wide range of tasks in the game. Your
 5257 advanced capabilities enable you to process and interpret gameplay
 5258 screenshots and other relevant information.
 5259
 5260 <\$image_introduction\$>
 5261
 5261 Task overview:
 5262 <\$task_description\$>
 5263
 5263 Current subtask:
 5264 <\$subtask_description\$>
 5265
 5266 Description: Please analyze and describe the screenshot image in a grid-
 5267 by-grid format from left to right and top to bottom and then provide
 5268 an overall image description. Pay attention to anything related to
 5269 the current subtask. The image is divided into a 5x3 grid, each cell
 5270 having its own coordinates. For each grid cell, describe the contents
 5271 in detail, focusing on any critical icons, or objects present in
 5272 that particular segment. If there are specific features such as
 5273 characters or text, mention these as well. After completing the
 5274 description for one cell, proceed to the next, for example, 'In grid
 5275 (1,1), [description]. In grid (2,1), [description].' and so on until
 5276 the entire image is covered.
 5276
 5276 Date_time: The date and time information in the game are shown on the
 5277 upper-right of the screenshot, in grid (5, 1). An example of the date
 5278 and time information is "Wed 10, 5:10 pm".
 5279
 5279 Energy: The current energy remains for the character doing actions. The
 5280 energy bar is shown on the bottom-right of the screenshot, in grid
 5281 (5, 3). The full energy is 270. An example of the energy information
 5282 is "150/270".
 5283
 5283 Weather: The current weather information in the game, the weather is one
 5284 from "Sunny", "Rainy", "Windy", "Snowy", "Stormy", "Festival", "
 5285 Wedding", and "null". If none of them applies, only output "null".
 5286
 5286 Dialog: If there are some dialogs shown in the screenshot, extract the
 5287 text of the conversation, like "Shopkeeper: What do you want to buy
 5288 ?", otherwise, only output "null".
 5289
 5289 Other: Other information that does not belong to the above categories. If
 5290 none of them applies, only output "null".
 5291

5292 You should only respond in the format described below and not output
 5293 comments or other information.
 5294 Description:
 5295 In grid (1,1), ...In grid (2,1), ...In grid (3,1), ...In grid (5,3), ...
 5296 Overall, the image shows...
 5297 Date_time:
 5298 Date and time information
 5299 Energy:
 5300 The number of energy remains showing in the energy bar
 5301 Weather:
 5302 Weather information
 5303 Dialog:
 5304 Dialog text
 5305 Other:
 5306 Other information is ...

5306 **Prompt 19: Stardew: Self-Reflection Shopping prompt.**

5307 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
 5308 the PC, equipped to handle a wide range of tasks in the game. Your
 5309 advanced capabilities enable you to process and interpret gameplay
 5310 screenshots and other relevant information. Your task is to examine
 5311 these inputs, interpret the in-game context, and determine whether
 5312 the executed action takes effect.
 5313 Target task:
 5314 <\$task_description\$>
 5315
 5316 Current subtask for completing the target task:
 5317 <\$subtask_description\$>
 5318
 5319 The reasoning for proposing the current subtask:
 5320 <\$subtask_reasoning\$>
 5321
 5322 Last executed action for completing the subtask:
 5323 <\$previous_action\$>
 5324
 5325 Reasoning for the last action:
 5326 <\$previous_reasoning\$>
 5327
 5328 Current Image description:
 5329 <\$image_description\$>
 5330
 5331 Toolbar information
 5332 <\$toolbar_information\$>
 5333
 5334 Summarization of recent history:
 5335 <\$history_summary\$>
 5336
 5337 <\$image_introduction\$>
 5338
 5339 Reasoning: You need to answer the following questions step by step to get
 5340 some reasoning based on the last action and sequential frames of the
 5341 character during the execution of the last action.
 5342 1. Are the characters' positions in these frames identical?
 5343 2. What is the executed action? Please answer this question not based on
 5344 the sequential frames.
 5345 3. Was the executed action successful? Give reasons. You should refer to
 the following rules:
 - Analyze by observing given sequential frames for detailed information.
 - If the action involves moving forward, it is considered unsuccessful
 only when the character's position remains unchanged across
 sequential frames, regardless of background elements and other people
 .
 - If you are not 100% sure that the action fails, regard it as success.

5346 4. If the last action is not executed successfully, what is the most
5347 probable cause? You should give only one cause and refer to the
5348 following rules:
5349 - The reasoning for the last action could be wrong.
5350 - If it is an interaction action such as `buy_item` or `do_action`, the most
5351 probable cause was that the action was unavailable at the current
5352 place, then you should move to a new place.
5353 - If it is a movement action, the most probable cause was that you were
5354 blocked by seen or unseen obstacles.
5355 - If there is an error report, analyze the cause based on the report.
5356 5. If the current subtask involves determining whether to enter the store
5357 , you need to compare the scene in the current screenshot with the
5358 scene in the screenshot from Memory to determine whether the
5359 character has entered the store, if not, then the task of entering
5360 the store is not complete.
5361 6. Is the subtask completed? Give your reasons. If you want to make any
5362 confirmation, regard it as a success. You should observe given
5363 sequential frames, do not rely on the text information.
5364 7. Is the target task completed? Give your reasons.
5365 8. If the current subtask involves purchase something, you should check
5366 the toolbar or purchase menu to see if the purchase was successful.
5367 Do not overbuy or miss the purchase.
5368 9. Do you think the subtask is reasonable? Give your reasons.
5369
5370 You should only respond in the format as described below.
5371 Reasoning:
5372 1. ...
5373 2. ...
5374 3. ...
5375 ...

Prompt 20: Stardew: Task Inference Shopping prompt.

5374 Assume you are a helpful AI assistant integrated with 'Stardew Valley' on
5375 the PC, equipped to handle a wide range of tasks in the game. You
5376 will also be given a summary of the history that happened before the
5377 last screenshot. You should assist in summarizing the events for
5378 future decision-making and also propose a new subtask, which is the
5379 most suitable subtask for the current situation, given the target
5380 task.
5381 Here is some helpful information to help you do the summarization and
5382 propose the subtask.
5383 Current task:
5384 <\$task_description\$>
5385 Previous proposed subtask for the task:
5386 <\$subtask_description\$>
5387
5388 Previous reasoning for proposing the subtask:
5389 <\$subtask_reasoning\$>
5390
5391 <\$image_introduction\$>
5392
5393 Current Image description:
5394 <\$image_description\$>
5395
5396 Last executed action:
5397 <\$previous_action\$>
5398
5399 Decision-making reasoning for the last executed action:
5400 <\$previous_reasoning\$>
5401
5402 Self-reflection for the last executed action:

5400 <\$self_reflection_reasoning\$>
5401
5402 The following is the summary of history that happened before the last
5403 screenshot:
5404 <\$previous_summarization\$>
5405 History_summary: Summarize what happened in the past experience,
5406 especially the last step according to the decision-making reasoning
5407 and self-reflection reasoning for the last executed action. The
5408 summarization needs to be precise, concrete and highly related to the
5409 task and follow the rules below.
5410 1. Summarize the tasks from the history and the current task. What is the
5411 current progress of the task? For example, to harvest a seed, you
5412 need to water the seed for 4 days. And you have already planted the
5413 seed and watered it for two days.
5414 2. Record the successful actions and organize them into events day by day
5415 .
5416 3. Do not forget the information and key events in the previous days.
5417 Subtask_reasoning: Decide whether the previous subtask is finished and
5418 whether it is necessary to propose a new subtask. The subtask should
5419 be straightforward, contribute to the target task and be most
5420 suitable for the current situation, which should be completed within
5421 a few actions. You should respond to me with:
5422 1. How to finish the target task? You should analyze it step by step.
5423 2. What is the current progress of the target task according to the
5424 analysis in step 1? Please do not make any assumptions if they are
5425 not mentioned in the above information. You should assume that you
5426 are doing the task from scratch.
5427 3. What is the previous subtask? Does the previous subtask finish? If so,
5428 give evidence that the task was completed. Or is it improper for the
5429 current situation? Then select a new one, otherwise you should reuse
5430 the last subtask.
5431 4. If you want to propose a new subtask, give reasons why it is more
5432 feasible for the current situation.
5433 5. The proposed subtask needs to be precise and concrete within one
5434 sentence. It should not be related to any skills.
5435 6. Do not mention any grid information in the subtask description.
5436 7. If the character does not reach the target place, you should propose a
5437 movement task to make him closer to the target.
5438 8. If you want to purchase items, then you should move up to stand in
5439 front of the shopkeeper's counter, move slightly to align with the
5440 green counter and buy items. After purchasing, you can move down to
5441 the exit and leave store.
5442 9. If you want to leave town, you should move along gray cobblestone road
5443 to the left of the store and the clinic.
5444 You should only respond in the format described below, and you should not
5445 output comments or other information.
5446 History_summary:
5447 The summary is...
5448 Subtask_reasoning:
5449 1. ...
5450 2. ...
5451 ...
5452 Subtask:
5453 The current subtask is

5449 **Prompt 21: Stardew: Action Planning Shopping prompt.**

5450 You are a helpful AI assistant integrated with 'Stardew Valley' on the PC
5451 , equipped to handle various tasks in the game. Your advanced
5452 capabilities enable you to process and interpret gameplay screenshots
5453 and other relevant information. By analyzing these inputs, you gain
a comprehensive understanding of the current context and situation

5454 within the game. Utilizing this insight, you are tasked with
5455 identifying the most suitable in-game action to take next, given the
5456 current task. You control the game character and can execute actions
5457 from the available action set. Upon evaluating the provided
5458 information, your role is to articulate the precise action you would
5459 deploy, considering the game's present circumstances, and specify any
5460 necessary parameters for implementing that action.

5461 Here is some helpful information to help you make the decision.

5462 Current subtask:
5463 <\$subtask_description\$>
5464

5465 Image description:
5466 <\$image_description\$>
5467

5468 Last executed action:
5469 <\$previous_action\$>

5470 Reasoning for the last action:
5471 <\$previous_reasoning\$>
5472

5473 Self-reflection for the last executed action:
5474 <\$previous_self_reflection_reasoning\$>

5475 Summarization of recent history:
5476 <\$history_summary\$>
5477

5478 Valid action set in Python format to select the next action:
5479 <\$skill_library\$>

5480 Grid System Information:
5481 1. Each grid has a coordinate (x,y). A larger x means that the grid is on
5482 the more eastern(right) side, and a larger y means that the grid is
5483 on the more southern(down) side. For example, moving from grid (1,3)
5484 to grid (1,1) requires move_up(duration=2) and moving from grid (1,1)
5485 to grid (2,1) requires move_right(duration=1)
5486 2. The larger the difference between the coordinates of the two grids,
5487 the longer it takes to move. Moving from grid (2,5) to grid (2,3)
5488 takes longer than moving from grid (2,3) to grid (1,3).
5489 <\$image_introduction\$>

5490 Based on the above information, analyze the current situation and provide
5491 the reasoning for what you should do for the next step to complete
5492 the task. Then, you should output the exact action you want to
5493 execute in the game. You should respond to me with:

5494 Reasoning: You should think step by step and provide detailed reasoning
5495 to determine the next action executed on the current state of the
5496 task. You need to answer the following questions step by step. You
5497 cannot miss the last question:
5498 1. Does the character already reach the target place? You must move
5499 close enough to the object to be in contact with it in order to
5500 interact with it. Just in the same grid with the target is not enough
5501 .
5502 2. Make use of the above image description, grid system information
5503 and current screenshot. Analyze whether the character has reached the
5504 target place. You must move close enough to the object to be in
5505 contact with it in order to interact with it. Just in the same grid
5506 with the target is not enough.
5507 3. What was the previous action? If the previous action was a
5508 movement, were you blocked?
5509 4. This is the most critical question. Based on the action rules and
5510 self-reflection, what should be the most suitable action in the valid

5508 action set for the next step? You should analyze the effects of the
5509 action step by step. You should not repeat the previous action again
5510 except for the movement action. Do not try to verify whether the
5511 previous action succeeded.

- 5512 5. Is the selected action the same as the last executed action? If
5513 yes, regenerate the action and give the reasons.
- 5514 6. Do all the selected actions exist in the valid action set? If no,
5515 regenerate the action and give the reasons.
- 5516 7. Where is the player's character? Notice that the player's
5517 character is a brown-haired man wearing a blue jacket.
- 5518 8. Does the selected action contribute to the current subtask?
- 5519 9. Analyze whether the selected action meets the requirements of the
5520 Actions below one by one. Does the generated action meet all the
5521 requirements? If not, regenerate the action and give the reasons.

5521 Actions: The requirements that the generated action needs to follow. The
5522 best action, or short sequence of actions without gaps, to execute
5523 next to progress in achieving the goal. Pay attention to the names of
5524 the available skills and to the previous skills already executed, if
5525 any. You should also pay more attention to the following action
5526 rules:

- 5527 1. You should output actions in Python code format and specify any
5528 necessary parameters to execute that action. If the function has
5529 parameters, you should also include their names and decide their
5530 values, like "move_right(duration=1)". If it does not have a
5531 parameter, just output the action, like "open_map()".
- 5532 2. You can only output at most two actions in the output.
- 5533 3. In the screenshots, the blue band represents the left side and the
5534 yellow band represents the right side. Please ignore character's
5535 facing direction and output the action in an absolute direction like
5536 right and left.
- 5537 4. If upon self-reflection you think the last action was unavailable
5538 at the current place, you MUST move to another place. Please do not
5539 try to execute the same action again.
- 5540 5. If you want to get out of the house, just use the skill
5541 go_through_door. You MUST NOT output any movement action behind this
5542 skill. And if the last executed action already contains this skill,
5543 do not execute this skill for the current step again.
- 5544 6. If upon self-reflection you think you were blocked, you MUST
5545 change the direction of moving, so that you can pass obstacles.
- 5546 7. You MUST NOT repeat the previous action again if you think the
5547 previous action fails.
- 5548 8. Your action should be strictly follow the analyze in the reasoning
5549 . Do not output any additional action not mentioned in the reasoning.
- 5550 9. If the current subtask includes purchasing items, here are some
5551 useful tips for you:
 - 5552 - Pierre's store is east of the character's house.
 - 5553 - if you do not see the store, you can move for a longer time each
5554 time, such move_right(duration=5). You can also move more distance to
5555 the left each time to get home faster.
 - 5556 - To successfully enable the purchase transaction, you should stand
5557 directly in front of the green counter, which left to the white
5558 counter with word 'for sale'.
 - 5559 - After aligning with green counter, you should purchase items.
 - 5560 - It is not necessary to positioned very precisely. If you stand
5561 near the green counter, you can try to purchase items.
- 5561 10. If the current subtask includes exiting town and returning home,
5562 here are some useful tips for you:
 - 5563 - Character' house is west of Pierre's store.
 - 5564 - There is a long distance from home to the store, so each movement
5565 should take a long duration, such as move_left(duration=5).
 - 5566 - Don't stand in the grass, move up and away from the lawn.
 - 5567 - The exit to the town is on the west(left) of Pierre's store and
5568 clinic. You should move left along the stone road, which has a wooden

5562 fence below it. If you gets stuck, move up slightly to get over the
 5563 obstacle.
 5564 11. If you want to enter a building, you should use `go_through_door(
 5565 door="xxx_entrance")`; If you want to leave a building, you should use
 5566 `go_through_door(door="xxx_exit")`.
 5567 - You can use `go_through_door(door="store_entrance")` to enter the
 5568 store.
 5569 - You can use `go_through_door(door="store_exit")` to leave the store.
 5570 - You can use `go_through_door(door="home_entrance")` to enter your
 5571 house.
 5572 - You can use `go_through_door(door="home_exit")` to leave your house.
 5573 12. If you want align with the target, you MUST move slightly. Each
 5574 movement take only 0.1 seconds, such as `move_xxx(duration=0.1)`.
 5575 You should only respond in the format described below, and you should not
 5576 output comments or other information.
 5577 Reasoning:
 5578 1. ...
 5579 2. ...
 5580 3. ...
 5581 Actions:
 5582 ```python
 5583 ... action(args1=x,args2=y)
 5584 ```

5584 K.4 PROMPTS FOR DEALER'S LIFE 2

5586 Prompt 22: Dealer's Life 2: Information Gathering prompt.

5588 Assume you are a helpful AI assistant integrated with "Dealer's Life 2"
 5589 on the PC, equipped to handle a wide range of tasks in the game. Your
 5590 advanced capabilities enable you to process and interpret gameplay
 5591 screenshots and other relevant information.
 5592 <\$image_introduction\$>
 5593
 5594 Current task:
 5595 <\$task_description\$>
 5596
 5597 Description: Please analyze and describe the screenshot image in detail
 5598 and then provide an overall image description. Most importantly,
 5599 identify the current page type and any relevant information related
 5600 to the task. If there are specific features such as characters or
 5601 text, mention these as well.
 5602
 5603 Budget: Bank Balance is shown at the top right of the screenshot.
 5604
 5605 Other: Other information that does not belong to the above categories. If
 5606 none of them applies, only output "null".
 5607
 5608 You should only respond in the format described below and not output
 5609 comments or other information.
 5610 Description:
 5611 The image shows...
 5612 Budget:
 5613 The amount of budget
 5614 Other:
 5615 Other information is ...

5613 Prompt 23: Dealer's Life 2: Self Reflection prompt.

5614 Assume you are a helpful AI assistant integrated with "Dealer's Life 2"
 5615 on the PC, equipped to handle a wide range of tasks in the game. Your

5616 advanced capabilities enable you to process and interpret gameplay
5617 screenshots and other relevant information. Your task is to examine
5618 these inputs, interpret the in-game context, and determine whether
5619 the executed action takes effect.

5620 Target task:
5621 <\$task_description\$>
5622

5623 Current subtask for completing the target task:
5624 <\$subtask_description\$>

5625 The reasoning for proposing the current subtask:
5626 <\$subtask_reasoning\$>
5627

5628 Last executed action for completing the subtask:
5629 <\$actions\$>

5630 Reasoning for the last action:
5631 <\$decision_making_reasoning\$>
5632

5633 Current budget:
5634 <\$budget\$>

5635 Summarization of recent history:
5636 <\$history_summary\$>
5637

5638 <\$image_introduction\$>
5639

5640 Reasoning: You need to answer the following questions step by step to get
5641 some reasoning based on the last action and sequential frames of the
5642 character during the execution of the last action.

- 5643 1. What is the executed action? Please answer this question not based on
5644 the sequential frames.
- 5645 2. Was the executed action successful? Give reasons. You should refer to
5646 the following rules:
- 5647 - If you are not 100% sure that the action fails, regard it as success.
- 5648 3. If the last action is not executed successfully, what is the most
5649 probable cause? You should give only one cause and refer to the
5650 following rules:
- 5651 - The reasoning for the last action could be wrong.
- 5652 - If it is an interaction action, the most probable cause was that the
5653 action was unavailable at the current place, then you should move to
5654 a new place.
- 5655 - If it is a movement action, the most probable cause was that you were
5656 blocked by seen or unseen obstacles.
- 5657 - If there is an error report, analyze the cause based on the report.
- 5658 4. Is the subtask completed? Give your reasons. If you want to make any
5659 confirmation, regard it as a success.
- 5660 5. Is the target task completed? Give your reasons.
- 5661 6. Do you think the subtask is reasonable? Give your reasons.

5662

5663 Success: You need to output whether the last action was executed
5664 successfully or not.

- 5665 - If the last action is successful, you should only output 'True'.
- 5666 Otherwise, you should only output 'False'.

5667

5668 You should only respond in the format described below.
5669 Reasoning:
5670 1. ...
5671 2. ...
5672 3. ...
5673 Success:
5674 True
5675 ...

5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723

Prompt 24: Dealer's Life 2: Task Inference prompt.

Assume you are a helpful AI assistant integrated with 'DealersLife2' on the PC, equipped to handle a wide range of tasks in the game. You will also be given a summary of the history that happened before the last screenshot. You should assist in summarizing the events for future decision-making and also propose a new subtask, which is the most suitable subtask for the current situation, given the target task.

Here is some helpful information to help you do the summarization and propose the subtask.

Current task:
<\$task_description\$>

Previous proposed subtask for the task:
<\$subtask_description\$>

Previous reasoning for proposing the subtask:
<\$subtask_reasoning\$>

<\$image_introduction\$>

Current budget:
<\$budget\$>

Current population:
<\$population\$>

Last executed action:
<\$actions\$>

Decision-making reasoning for the last executed action:
<\$decision_making_reasoning\$>

Self-reflection for the last executed action:
<\$self_reflection_reasoning\$>

The following is the summary of history that happened before the last screenshot:
<\$previous_summarization\$>

History_summary: Summarize what happened in the past experience, especially the last step according to the decision-making reasoning and self-reflection reasoning for the last executed action. The summarization needs to be precise, concrete and highly related to the task and follow the rules below.

1. Summarize the tasks from the history and the current task. What is the current progress of the task?
2. Record the successful actions and organize them into events day by day
3. Do not forget the information and key events in the previous days.
4. If you are watering a seed. Record how many times you have watered and calculate how many days you have to water before you can harvest according to the toolbar information provided above.

Subtask_reasoning: Decide whether the previous subtask is finished and whether it is necessary to propose a new subtask. The subtask should be straightforward, contribute to the target task and be most suitable for the current situation, which should be completed within a few actions. You should respond to me with:

1. How to finish the target task? You should analyze it step by step.
2. What is the current progress of the target task according to the analysis in step 1? Please do not make any assumptions if they are

5724 not mentioned in the above information. You should assume that you
 5725 are doing the task from scratch.

- 5726 3. What is the previous subtask? Does the previous subtask finish? Or is
- 5727 it improper for the current situation? Then select a new one,
- 5728 otherwise you should reuse the last subtask.
- 5729 4. If you want to propose a new subtask, give reasons why it is more
- 5730 feasible for the current situation.
- 5731 5. The proposed subtask needs to be precise and concrete within one
- 5732 sentence. It should not be related to any skills.
- 5733 6. Do not mention any grid information in the subtask description.

5734 You should only respond in the format described below, and you should not
 5735 output comments or other information.

5736 History_summary:
 5737 The summary is ...
 5738 Subtask_reasoning:
 5739 1. ...
 5740 2. ...
 5741 3. ...
 5742 Subtask:
 5743 The current subtask is ...

5743 **Prompt 25: Dealer's Life 2: Action Planning prompt.**

5744 You are a helpful AI assistant integrated with "Dealer's Life 2" on the
 5745 PC, equipped to handle various tasks in the game. Your advanced
 5746 capabilities enable you to process and interpret gameplay screenshots
 5747 and other relevant information. By analyzing these inputs, you gain
 5748 a comprehensive understanding of the current context and situation
 5749 within the game. Utilizing this insight, you are tasked with
 5750 identifying the most suitable in-game action to take next, given the
 5751 current task. You control the game character and can execute actions
 5752 from the available action set. Upon evaluating the provided
 5753 information, your role is to articulate the precise action you would
 5754 deploy, considering the game's present circumstances, and specify any
 5755 necessary parameters for implementing that action.

5756 Here is some helpful information to help you make the decision.

5757 Current subtask:
 5758 <\$subtask_description\$>

5759 Current page type:
 5760 <\$coordinates\$>

5761 Current budget:
 5762 <\$budget\$>

5763 Last executed action:
 5764 <\$actions\$>

5765 Reasoning for the last action:
 5766 <\$decision_making_reasoning\$>

5767 Self-reflection for the last executed action:
 5768 <\$self_reflection_reasoning\$>

5769 Summarization of recent history:
 5770 <\$history_summary\$>

5771 Valid action set in Python format to select the next action:
 5772 <\$skill_library\$>

5773 <\$image_introduction\$>

5778 Based on the above information, analyze the current situation and provide
5779 the reasoning for what you should do for the next step to complete
5780 the task. Then, you should output the exact action you want to
5781 execute in the game. You should respond to me with:

5782 Reasoning: You should think step by step and provide detailed reasoning
5783 to determine the next action executed on the current state of the
5784 task. You need to answer the following questions step by step. You
5785 cannot miss the last question:

- 5786 1. Analyze the information in the screenshot. What can you observe in
5787 the screenshot? Please list some key elements.
- 5788 2. What is the current task? What are the requirements to achieve the
5789 goal?
- 5790 3. What have you done so far in the game? What are the results of the
5791 previous actions?
- 5792 4. What is your next step to achieve the goal? What is your plan? Why
5793 do you choose this action? Please explain the reasoning behind your
5794 decision.
- 5795 5. If you were to respond to the customer's dialogue on the dialogue
5796 page, which of the listed responses in the screenshot would you
5797 choose? Why?
- 5798 6. If you are to make an offer to a customer, how would you determine
5799 the price? You should determine the customer's role here. If the
5800 customer is a "seller", you should offer a price lower than the item's
5801 value. If the customer is a "buyer", you should offer a price
5802 higher than the item's value. Please explain your reasoning.
- 5803 7. If the customer rejects your offer and makes a counteroffer, what
5804 would you do? Would you accept the counteroffer or refuse the deal?
5805 Why?
- 5806 8. What does the current screen image show? is it a giving price page
5807 (it at least should show price \$ in the right bottom of the screen
5808 image) or a non-giving price page and why?

5809 Actions: The requirements that the generated action needs to follow. The
5810 best action, or short sequence of actions without gaps, to execute
5811 next to progress in achieving the goal. Pay attention to the names of
5812 the available skills and the previous skills already executed, if
5813 any. You should also pay more attention to the following action rules
5814 :

- 5815 1. You should output actions in Python code format and specify any
5816 necessary parameters to execute that action. If the function has
5817 parameters, you should also include their names and decide their
5818 values, like "move_right(duration=1)". If it does not have a
5819 parameter, just output the action, like "open_map()".
- 5820 2. Given the current situation and task, you should only choose the
5821 most suitable action from the valid action set. You cannot use
5822 actions that are not in the valid action set to control the character
5823 .
- 5824 3. In the screenshots, the blue band represents the left side and the
5825 yellow band represents the right side. Please ignore the character's
5826 facing direction and output the action in an absolute direction like
5827 right and left.
- 5828 4. If you want to run as a successful dealer in conversation with the
5829 customer, you should follow these rules:
 - 5830 4.1 Check the customer's dialogue.
 - 5831 - If the customer is introducing himself and his purpose of
visiting your shop, you should always respond with "Let's see" to
make them potential buyers. This will be the first option in the
dialogue and you should select it.
 - 4.2 Check the customer's response.
 - If the customer has shown you the details of the items and you
have completed by closing the item detail page, you should respond
with "Let's deal" to make an offer. This will be the first option in
the dialogue and you should select it.

5832 5. If you want to run as a successful dealer in making an offer and
 5833 deciding whether to take the offer or counteroffer, you should follow
 5834 these rules:
 5835 5.1 Check the customer's role.
 5836 - If the customer is a "seller", you should offer a price lower
 5837 than the item's value. You should also consider your budget.
 5838 - If the customer is a "buyer", you should offer a price higher
 5839 than the item's value.
 5840 5.2 Check the item's details.
 5841 - You should check the item's "rarity", "condition", and "estimate"
 5842 to determine the price you offer.
 5843 6. If you have opened up the buyer's or seller's character trait page
 5844 , you should call the function to close the description page to
 5845 proceed with the next action. You should NOT call any other skill
 5846 like dialogue().
 5847 7. Your action should strictly follow the analysis in the reasoning.
 5848 Do not output any additional action not mentioned in the reasoning.

5849 You should only respond in the format described below, and you should not
 5850 output comments or other information.
 5851 Reasoning:
 5852 1. ...
 5853 2. ...
 5854 3. ...
 5855 Actions:
 5856 ```python
 5857 action(args1=x,args2=y)
 5858 ```

5858 K.5 PROMPTS FOR SOFTWARE APPLICATIONS

5861 Prompt 26: Chrome: Information Gathering prompt.

5862 Assume you are a helpful AI assistant integrated with 'Google Chrome' on
 5863 the PC, equipped to handle a wide range of tasks in the application.
 5864 Your advanced capabilities enable you to process and interpret
 5865 application screenshots and other relevant information.

5866 Image introduction:
 5867 <\$image_introduction\$>

5868 Overall task:
 5869 <\$task_description\$>

5870 Subtask description:
 5871 <\$subtask_description\$>

5872 Image_Description:
 5873 1. Please describe the screenshot image in detail. Pay attention to any
 5874 details in the image, if any, especially critical icons, or created
 5875 items.
 5876 2. If the image includes a mouse cursor, please describe what UI element
 5877 the mouse is currently located near. Pay attention to the coordinates
 5878 of the pointer tip, not the center of the mouse cursor.
 5879 3. Pay attention to all UI items and contents in the image. Do not make
 5880 assumptions about the layout.

5881 Description_of_bounding_boxes:
 5882 Please provide a list of EVERY bounding box from label ID of 1 to <
 5883 \$length_of_som_map\$> ONE BY ONE. The label IDs are marked in the
 5884 upper left corner of the bounding boxes.
 5885 For bounding boxes containing text, provide ONLY the text.
 For bounding boxes without text, brief description of the function.

5886 Format your response as follows: '1: function_a', '2: text_b', ..., '<
 5887 \$length_of_som_map\$: function_b'. Don't write anything you are not
 5888 sure about.

5889

5890 Target_object_name: Assume you can use an object detection model to
 5891 detect the most relevant object or UI item for completing the current
 5892 task if needed. What item should be detected to complete the task
 5893 based on the current screenshot and the current task? You should obey
 5894 the following rules:

- 5894 1. Identify an item that is relevant to the current or intermediate
- 5895 target of the task. If the item is within a bounding box in the
- 5896 screenshot, please include the corresponding label ID.
- 5897 2. If no explicit item is specified, only output "null".
- 5898 3. If there is no need to detect an object, only output "null".

5899 Reasoning_of_object: Why was this object chosen, or why is there no need
 5900 to detect an object?

5901 You should only respond in the format described below and not output
 5902 comments or other information. DO NOT change the title of each item.

5903 Image_Description:

- 5904 1. ...
- 5905 2. ...
- 5906 3. ...

5907 Description_of_bounding_boxes:

5908 Format like: 1: function_a', '2: text_b', ..., '<\$len_of_bound_boxes\$:
 5909 function_b

5910

5911 Target_object_name:
 5912 label ID, Name

5913 Reasoning_of_object:
 5914 ...

5915

5916 **Prompt 27: Chrome: Self-Reflection prompt.**

5917 Assume you are a helpful AI assistant integrated with 'Google Chrome' on
 5918 the PC, equipped to handle a wide range of tasks in the application.
 5919 Your advanced capabilities enable you to process and interpret
 5920 application screenshots and other relevant information. Your task is
 5921 to examine these inputs, interpret the in-application and OS context,
 5922 and determine whether the executed action has taken the correct
 5923 effect.

5924 Overall task description:
 5925 <\$task_description\$>

5926 Image introduction:
 5927 <\$image_introduction\$>

5928

5929 Last executed action with parameters used:
 5930 <\$previous_action_call\$>

5931

5932 Implementation of the last executed action:
 5933 <\$action_code\$>

5934

5935 Error report for the last executed action:
 5936 <\$executing_action_error\$>

5937

5938 Key reason for the last action:
 5939 <\$key_reason_of_last_action\$>

5939

5939 History Summarization
 <\$history_summary\$>

5940
5941 Success_Detection flag for the overall task:
5942 <\$success_detection\$>
5943
5944 Valid action set in Python format to select the next action:
5945 <\$skill_library\$>
5946
5947 Current and previous screenshot are the same:
5948 <\$image_same_flag\$>
5949
5950 Mouse position in the current screenshot is the same as in the previous
5951 screenshot:
5952 <\$mouse_position_same_flag\$>
5953
5954 Self_Reflection_Reasoning:
5955 You need to answer the following questions, step by step, to describe
5956 your reasoning based on the history summarization, last action and
5957 sequential screenshots of the application during the execution of the
5958 last action.
5959 1. Please describe what the page is in the current screenshot. Respond in
5960 one sentence.
5961 2. What is the last executed action based on the text information above?
5962 3. Was the last executed action successful? Give reasons. You should
5963 refer to the following rules:
5964 - If the last action executed was empty, then the previous action is
5965 deemed successful.
5966 - If the action involves moving the mouse, it is considered unsuccessful
5967 when the mouse position remains unchanged or moves in an incorrect
5968 way across sequential screenshots, regardless of background elements
5969 and other items.
5970 - If the position to move the mouse to was incorrect and the mouse didn't
5971 reach the target UI element, pay more attention to the accurate
5972 coordinates to move to.
5973 - If the operation involves type text, it will be considered unsuccessful
5974 when the corresponding text does not appear in the diagram,
5975 regardless of background elements and other items.
5976 - If the action seemed to have no effect, pay attention to the latest
5977 mouse position. Did it move? Did it get closer to the target UI
5978 element? Where are the target coordinates in the action wrong? The
5979 position of the mouse cursor on the screenshot shows their location.
5980 - Was some unrelated UI item triggered by the last action?
5981 4. If the last action is not executed successfully, what is the most
5982 probable cause? You should give only one cause and refer to the
5983 following rules:
5984 - The reasoning for the last action could be wrong.
5985 - If it was an action involving moving the mouse or the text cursor, the
5986 most probable cause was that the coordinates used were incorrect.
5987 - If it is an interaction action, the most probable cause was that the
5988 action was unavailable or not activated in the current state.
5989 - If an unrelated change happened in the UI, the most probable cause was
5990 that the action triggered an incorrect UI element.
5991 - If there is an error report, analyze the cause based on the report.
5992
5993 Success_Detection:
Based on the history summarization, the last action, the current
screenshots and the Success_Detection flag, determine whether the
overall task "<\$task_description\$>" was successful. This assessment
should consider the overall task's success, not just individual
actions.
- If the last action executed was an empty list and "<\$success_detection\$
>" indicates the task is successful, then the overall task has a high
chance of being considered a success.
- If the overall task was unsuccessful, specify the reason of failure and
which steps are missing.
- If the overall task was successful, ONLY output "SUCCESSFUL".

5994

5995

You should only respond in the format as described below.

5996

Self_Reflection_Reasoning:

5997

1. ...

5998

2. ...

5999

3. ...

6000

Success_Detection:

6001

...

6002

6003

6004

Prompt 28: Chrome: Task Inference prompt.

6005

Assume you are a helpful AI assistant integrated with 'Google Chrome' on the the PC, equipped to handle a wide range of tasks in the game. You will be sequentially given <\$event_count\$> screenshots and corresponding descriptions of recent events. You will also be given a summary of the history that happened before the last screenshot. You should assist in summarizing the events for future decision-making and also in proposing the most suitable subtask to execute next, given the target task.

6012

6013

Here is some helpful information to help you do the summarization and propose the subtask.

6014

6015

Overall task description:

6016

<\$task_description\$>

6017

6018

Previous proposed subtask for the task:

6019

<\$subtask_description\$>

6020

6021

Previous reasoning for proposing the subtask:

6022

<\$subtask_reasoning\$>

6023

6024

Image introduction:

<\$image_introduction\$>

6025

6026

Last executed action:

<\$previous_action\$>

6027

6028

Error report for the last executed action:

6029

<\$executing_action_error\$>

6030

6031

Key decision-making reasoning for the last executed action:

6032

<\$previous_reasoning\$>

6033

6034

Self-reflection for the last executed action:

<\$self_reflection_reasoning\$>

6035

6036

Success_Detection for the overall task:

6037

<\$success_detection\$>

6038

6039

The following is the summary of history that happened before the last screenshot:

6040

<\$previous_summarization\$>

6041

6042

History_summary: Summarize what happened in the past experience, especially the last step according to the decision-making reasoning and self-reflection reasoning for the last executed action. The summarization needs to be precise, concrete, highly related to the task, and follow the rules below.

6043

6044

6045

6046

6047

1. Determine if the task has been completed successfully. If it is successful, ignore question 2 to 5.
2. Summarize the tasks from the history and the current task. What is the current progress of the task? For example, to open a file, you

6048 first need to select the file, then open it by clicking somewhere or
6049 using the keyboard. Subtasks may have other pre-requisites.
6050 3. Record the successful actions and organize them into events, step
6051 by step.
6052 4. Which subtask has been completed? Which subtasks have not? Do not
6053 forget the information and key events in the previous steps of the
6054 overall task.

6055 Subtask_reasoning: Decide whether the previous subtask is finished and
6056 whether it is necessary to propose a new subtask. The subtask should
6057 be straightforward, contribute to the target task, and be most
6058 suitable for the current situation; which should be completed within
6059 a few actions. You should respond with the following item.
6060 1. Think about a hotkey related to the overall task and next subtask,
6061 please specify what it is.
6062 2. Based on the current screenshot, identify the most direct and
6063 easiest way to complete the task.
6064 3. Analyze the target task step by step to determine how to complete
6065 it.
6066 4. What is the previous subtask? Has the previous subtask finished
6067 due to self-reflection? Or is it improper for the current situation?
6068 If finished or improper, please select a new one, otherwise you
6069 should reuse the last subtask.
6070 5. If you want to propose a new subtask, give reasons why it is more
6071 feasible for the current situation. Please strictly follow the
6072 description and requirements in the current task.
6073 6. The proposed subtask needs to be precise and concrete within one
6074 sentence. It should not be directly related to any skills.

6075 You should only respond in the format described below, and you should not
6076 output comments or other information.

6077 History_summary:
6078 1. ...
6079 2. ...
6080 ...

6081 Subtask_reasoning:
6082 1. ...
6083 2. ...
6084 ...

6085 Subtask_description:
6086 The current subtask is ...

Prompt 29: Chrome: Action Planning prompt.

6087 You are a helpful AI assistant integrated with 'Google Chrome' on the PC,
6088 equipped to handle a wide range of tasks in the application. Your
6089 advanced capabilities enable you to process and interpret application
6090 screenshots and other relevant information. By analyzing these
6091 inputs, you gain a comprehensive understanding of the current context
6092 and situation within the application. Utilizing these insights, you
6093 are tasked with identifying the most suitable in-application action
6094 to take next, given the current task. You control the application and
6095 can execute actions from the available action set to manipulate its
6096 UI. Upon evaluating the provided information, your role is to
6097 articulate the precise actions you should perform, considering the
6098 application's present circumstances, and specify any necessary
6099 parameters for implementing that action.
6100 Here is some helpful information to help you make the decision.

6101 Overall task description:
<\$task_description\$>

6102 Subtask description:
6103 <\${subtask_description}>
6104

6105 Few shots:
6106 <\${few_shots}>

6107 Image introduction:
6108 <\${image_introduction}>
6109

6110 Current and previous screenshot are the same:
6111 <\${image_same_flag}>

6112 Mouse position in the current screenshot is the same as in the previous
6113 screenshot:
6114 <\${mouse_position_same_flag}>

6115

6116 Description of current screenshot:
6117 <\${image_description}>

6118

6119 Description of label IDs:
6120 <\${description_of_bounding_boxes}>

6121

6122 Last executed action:
6123 <\${previous_action}>

6124

6125 Key reason for the last action:
6126 <\${key_reason_of_last_action}>

6127

6128 Self-reflection for the last executed action:
6129 <\${previous_self_reflection_reasoning}>

6130

6131 Summarization of recent history:
6132 <\${previous_summarization}>

6133

6134 Valid action set in Python format to select the next action:
6135 <\${skill_library}>

6136

6137 Success detection for overall task:
6138 <\${success_detection}>

6139

6140 Based on the above information, you should first analyze the current
6141 situation and provide the reasoning for what you should do for the
6142 next step to complete the task. Then, you should output the exact
6143 action you want to execute in the application.

6144

6145 Pay attention to all UI items and contents in the image. DO NOT make
6146 assumptions about the layout! If the image includes a mouse cursor,
6147 pay close attention to the coordinates of the pointer tip, not the
6148 centre of the mouse cursor.

6149

6150 You should respond to me with the following information, and you MUST
6151 respond one by one.

6152

6153 Decision_Making_Reasoning: You should think step by step and provide
6154 detailed reasoning to determine the next action executed on the
6155 current state of the task.

1. Does "<\${success_detection}>" mean the overall task was successful?
If successful, ignore questions 2 to 12.
2. Which skill in the Skill Library "<\${skill_library}>" has the
closest semantics to the current subtask "<\${subtask_description}>"?
If there is an answer, select it as the output action.
3. Prefer keyboard operation instead of mouse operation. Are there
any keyboard actions, such as using shortcut keys or pressing "enter
", to finish the current step or overall task? If there is, please
specify which it is.
4. Based on the action rules, self-reflection and previous
summarization, what should be the most suitable action in the valid

6156 action set for the next step? You should analyze the effects of the
6157 action step by step.

6158 5. If the previous action is unsuccessful, DO NOT repeat the previous
6159 action, consider an alternative action if possible. If there is an
6160 alternative action, please specify what it is, such as clicking
6161 different label IDs or using different shortcut keys.

6162 6. Always try pressing "enter" first instead of clicking it with the
6163 mouse, if the button you want to click is active.

6164 7. Check whether the UI element you want to operate exists in the
6165 current screenshot. If not, you can choose to return to the previous
6166 page or reopen a tab.

6167 8. In the current screenshot, identify the label ID of the bounding
6168 box most relevant to the current step. If there is text within this
6169 bounding box, please provide the text.

6170 9. If mouse actions are necessary, use that specific bounding box
6171 label ID (if shown in the current screenshot) as a parameter, rather
6172 than directly generating normalized x and y coordinates. If there is
6173 any relevant label ID, please specify which it is.

6174 10. If a dialog box appears, make sure to check the content of the
6175 dialog box to determine if the task is complete. For instance, when a
6176 download dialog box appears, the task is only completed after
6177 pressing the Enter key or clicking "Save".

6178 11. If you need to use an action outside an open menu or dialog box,
6179 please close the current menu or dialog box before trying the next
6180 action.

6181 12. If you anticipate that the next step involves typing text,
6182 confirm that the last executed action was a click at the appropriate
6183 input box. If not, it is mandatory to click on the corresponding
6184 input box before proceeding with typing.

6185 Actions: The best action, or short sequence of actions without gaps, to
6186 execute next to progress in achieving the goal. Pay attention to the
6187 names of the available skills and the previous skills already
6188 executed, if any. Pay special attention to the coordinates of any
6189 action that needs them. Do not make assumptions about the location of
6190 UI elements or their coordinates, analyse in detail any provided
6191 images. You should also pay more attention to the following action
6192 rules:

6193 1. If "<\$success_detection\$>" means the overall task was successful
6194 or equal to "True", then the output action MUST be empty like ''. Be
6195 careful to check the task was really successful.

6196 2. You should output actions in Python code format and specify any
6197 necessary parameters to execute that action. Only use function names
6198 and argument names exactly as shown in the valid action set. If a
6199 function has parameters, you should also include their names and
6200 decide their values, like "press_shift(duration=1)". If it does not
6201 have a parameter, just output the action, like "release_mouse_buttons
6202 ()".

6203 3. Before typing text, ensure that the last executed action involved
6204 clicking on the relevant input box. If the last action was not a
6205 click on this input box, the required action MUST be to click on the
6206 corresponding input box before proceeding.

6207 4. Given the current situation and task, you should only choose the
6208 most suitable action from the valid action set. You cannot use
6209 actions that are not in the valid action set to control the
6210 application.

6211 5. When you perform a mouse action, always select the target UI
6212 element closest to the UI element of the previous action for
6213 operation.

6214 6. When you decide to operate on a file, such as downloading it,
6215 please pay attention to the path and name of the current file.

6216 Key_reason_of_last_action: Summarize the key reasons why you output this
6217 action.

6210 You should only respond in the format described below. In your reasoning
 6211 for the chosen actions, also describe which item you decided to
 6212 interact with and why. DO NOT change the title of each item. You
 6213 should not output other comments or information besides the format
 6214 below.

6215 Decision_Making_Reasoning:
 6216 1. ...
 6217 2. ...
 6218 3. ...
 6219 ...

6219 Actions:
 6220 ```python
 6221 action(args1=x, args2=y)
 6222 ```

6223 Key_reason_of_last_action:
 6224 ...
 6225

6226
 6227 **Prompt 30: Outlook: Information Gathering prompt.**

6228 You an expert helpful AI assistant which follows instructions and
 6229 performs desktop computer tasks as instructed. You have expert
 6230 knowledge of 'Microsoft Outlook' on the PC and can handle a wide
 6231 range of tasks in the application using the keyboard, shortcut keys,
 6232 and mouse operations. For each step, you will get one or more
 6233 observation images, which are screenshots of the computer screen.
 6234 Your advanced capabilities enable you to process and interpret these
 6235 application screenshots and other relevant information in detail. The
 6236 screenshots include numerical tags (label IDs) and bounding boxes
 marking some UI items.

6237 Image introduction:
 6238 <\$image_introduction\$>

6239 Overall task:
 6240 <\$task_description\$>

6241 Subtask description:
 6242 <\$subtask_description\$>

6243 Image_Description:
 6244 1. Please describe the screenshot image in detail. Pay attention to any
 6245 details in the image, if any, especially critical icons, open menus
 6246 or dialogs, and any instructions for the application user. Focus on
 6247 the image contents and the situation in the application.
 6248 2. If the image includes a mouse cursor, please describe what UI element
 6249 the mouse is currently located near. Pay attention to the coordinates
 6250 of the pointer tip, not the center of the mouse cursor.
 6251 3. Pay attention to all UI items and contents in the image. Do not make
 6252 assumptions about the layout.
 6253 4. DO NOT describe overlaid bounding boxes in this description, only the
 6254 relevant UI items themselves. Focus on the state of the application
 6255 UI and what the key UI items of interest for the task would be.
 6256 Describe any relevant open panels, dialogs, menus, etc.

6257 Target_object_name:
 6258 As an application expert and a helpful assistant, you can determine the
 6259 most relevant UI items for completing the current subtask, if needed.
 6260 What item should be detected to complete the task based on the
 6261 current screenshot and the current subtask? You should obey the
 6262 following rules:
 6263 1. The item should be present in the screen and relevant to the current
 subtask or overall task. Just name the item, without any modifiers or
 extra information.

6264 2. If the item of interest is not on the current screen, only output "
 6265 Target items not in current screen".
 6266 2. If no explicit item is specified, only output "null".
 6267 3. If there is no need to detect a target item in this state, only output
 6268 "null". You must output this field in the response.

6269 Reasoning_of_object: Why was this item chosen, or why is there no need to
 6270 detect an UI item at this stage?
 6271

6272 You should only respond in the format described below and not output
 6273 comments or other information. DO NOT change the titles of any
 6274 response items.

6275 Image_Description:
 6276 1. ...
 6277 2. ...
 6278 3. ...

6279 Target_object_name:
 6280 name
 6281

6282 Reasoning_of_object:
 6283 ...

6284

6285 **Prompt 31: Outlook: Self-Reflection prompt.**

6286 You are an expert helpful AI assistant which follows instructions and
 6287 performs desktop computer tasks as instructed. You have expert
 6288 knowledge of 'Microsoft Outlook' on the PC and can handle a wide
 6289 range of tasks in the application using the keyboard, shortcut keys,
 6290 and mouse operations. For each step, you will get one or more
 6291 observation images, which are screenshots of the computer screen.
 6292 Your advanced capabilities enable you to process and interpret these
 6293 application screenshots and other relevant information in detail.
 6294 You MUST examine all inputs, interpret the in-application and OS contexts
 6295 , and determine whether the executed action has taken the correct
 effect.

6296 Overall task description:
 6297 <\$task_description\$>

6298 Execution step images:
 6299 <\$image_introduction\$>
 6300

6301 Current image description:
 6302 <\$current_image_description\$>
 6303

6304 Last executed action with parameters used:
 6305 <\$previous_action_call\$>

6306 Implementation of the last executed action:
 6307 <\$action_code\$>
 6308

6309 Error report for the last executed action:
 6310 <\$executing_action_error\$>

6311 Key reason for the last action:
 6312 <\$key_reason_of_last_action\$>
 6313

6314 Success_Detection flag for the overall task:
 6315 <\$success_detection\$>

6316 Valid action set in Python format to select the next action:
 6317 <\$skill_library\$>

6318 Current and previous screenshot are the same:
6319 <\$image_same_flag\$>
6320

6321 Mouse position in the current screenshot is the same as in the previous
6322 screenshot:
6323 <\$mouse_position_same_flag\$>

6324 As the textual history may not completely record some effects of previous
6325 actions, you should closely evaluate every part of the screenshots
6326 to understand what was supposed to happen and what has actually
6327 happened.

6328 Self_Reflection_Reasoning: You need to answer the following questions,
6329 step by step, to describe your reasoning based on the last action and
6330 sequential screenshots of the application during the execution of
6331 the last action. Any action involving x and y coordinates is an
6332 action involving movement.

- 6333 1. What is the last executed action not based on the sequential
6334 screenshots?
- 6335 2. Was the last executed action successful? Give reasons. You should
6336 refer to the following rules:
 - 6337 - If the action involved typing text, was it typed correctly at the right
6338 location? Do not trust only the textual information as it may not
6339 provide enough detail. Perform a thorough and detailed inspection of
6340 the provided screenshots! This is a critical check at every step!
 - 6341 - If the action involved moving the mouse, it is considered unsuccessful
6342 when the mouse position remains unchanged or moved in an incorrect
6343 way across sequential screenshots, regardless of background elements
6344 and other items.
 - 6345 - If the position to move the mouse to was incorrect and the mouse didn't
6346 reach the target UI element, pay more attention to the accurate
6347 location or UI item to move to.
 - 6348 - Are you sure the latest screenshot shows UI items that correspond to
6349 the success of the previous action? For example, if you tried to
6350 click on the "Junk" folder, the latest screenshot should show that
6351 folder, not "Inbox" or others.
 - 6352 - Triggering an action in the last step is not enough to say it was
6353 completely successfully. At least some relevant UI must change. Pay
6354 attention to the application states in the screenshots and any
6355 differences.
 - 6356 - If the action seemed to have no effect, pay attention to the latest
6357 mouse position. Did it move? Did it get closer to the target UI
6358 element? Was the target in the action wrong? The position of the
6359 mouse cursor on the screenshot shows their location.
- 6360 3. If the last action is not executed successfully, what is the most
6361 probable cause? You should give only one cause and refer to the
6362 following rules:
 - 6363 - The reasoning for the last action could be wrong.
 - 6364 - If it was an action involving moving the mouse or the text cursor, the
6365 most probable cause was that the coordinates or destination location
6366 used were incorrect.
 - 6367 - If you already tried the same action more than one time and there was
6368 no effect. DO NOT REPEAT the same action again until you have tried
6369 something else.
 - 6370 - If it is an interaction action, the most probable cause was that the
6371 action was unavailable or not activated at the current state.
 - If an unrelated change happened in the UI, the most probable cause was
that the action triggered an incorrect UI element.
 - If there is any error report, analyze the cause based on the report.

6369 Success_Detection:
6370 Based on the last action, the current screenshots and the
6371 Success_Detection flag, determine whether the overall task was

6372 successful. This assessment should consider the overall task's
 6373 success, not just individual actions.
 6374 - If the task was unsuccessful, specify the reason of failure and which
 6375 steps are missing.
 6376 - Pay extra attention to the application state in the latest screenshot.
 6377 Is it consistent with the task being completed successfully? Or is
 6378 there evidence that the task is still ongoing?
 6379 - If the task was successful, ONLY output "SUCCESSFUL".

6380 You should only respond in the format as described below.
 6381 Self_Reflection_Reasoning:
 6382 1. ...
 6383 2. ...
 6384 3. ...

6385 Success_Detection:
 6386 ...

6387
 6388
 6389 **Prompt 32: Outlook: Task Inference prompt.**

6390 You are an expert helpful AI assistant which follows instructions and
 6391 performs desktop computer tasks as instructed. You have expert
 6392 knowledge of 'Microsoft Outlook' on the PC and can handle a wide
 6393 range of tasks in the application using the keyboard, shortcut keys,
 6394 and mouse operations. For each step, you will get one or more
 6395 observation images, which are screenshots of the computer screen.
 6396 Your advanced capabilities enable you to process and interpret these
 6397 application screenshots and other relevant information in detail.
 6398 You will receive a sequence of <\$event_count\$> screenshots, corresponding
 6399 descriptions of recent events, and a summary of the history of
 6400 events before the last screenshot. Please summarize the events for
 6401 future decision-making and also propose the most suitable subtasks to
 6402 execute next, given the overall target task.

6403 Here is some helpful information to help you do the summarization and
 6404 propose the subtask.

6405 Overall task description:
 6406 <\$task_description\$>

6407 Previous proposed subtask for the task:
 6408 <\$subtask_description\$>

6409 Previous reasoning for proposing the subtask:
 6410 <\$subtask_reasoning\$>

6411 Image introduction:
 6412 <\$image_introduction\$>

6413 Last executed action:
 6414 <\$previous_action\$>

6415 Error report for the last executed action:
 6416 <\$executing_action_error\$>

6417 Key decision-making reasoning for the last executed action:
 6418 <\$previous_reasoning\$>

6419 Self-reflection for the last executed action:
 6420 <\$self_reflection_reasoning\$>

6421 Success_Detection for the overall task:
 6422 <\$success_detection\$>

6426 The following is the summary of history that happened before the last
6427 screenshot:
6428 <\$previous_summarization\$>
6429
6430 History_summary: Summarize what happened in the past experience,
6431 especially the last step according to the decision-making reasoning
6432 and self-reflection reasoning for the last executed action. The
6433 summarization needs to be precise, concrete, highly related to the
6434 task, and follow the rules below.
6435 1. Summarize the tasks from the history and the current task. What is the
6436 current progress of the task? For example, to open a file, you first
6437 need to select the file, then open it by clicking somewhere or using
6438 the keyboard. Subtasks may have other pre-requisites.
6439 2. Record the successful actions and organize them into events, step by
6440 step.
6441 3. Which subtask has been completed? Which subtasks have not?
6442 4. Do not forget the information and key events in the previous steps of
6443 the overall task.
6444
6445 Subtask_reasoning: Decide whether the previous subtask is finished and
6446 whether it is necessary to propose a new subtask. The subtask should
6447 be straightforward, contribute to the target task, and be most
6448 suitable for the current situation; which should be completed within
6449 a few actions. Use your knowledge of keyboard shortcuts to accomplish
6450 subtasks. You should respond with:
6451 1. How to finish the target task? You should analyze it step by step.
6452 Subtasks can involve keyboard shortcuts, using the mouse, or
6453 executing other skills.
6454 2. What is the current progress of the target task according to the
6455 analysis in question 1? Please do not make any assumptions if needed
6456 information is not mentioned previously. You should assume that you
6457 are doing the task from scratch. Please strictly follow the
6458 description and requirements in the current overall task.
6459 3. What is the previous subtask? Has the previous subtask finished
6460 according to self-reflection? Or is it improper for the current
6461 situation? If the last subtask already finished or now is improper,
6462 please select a new one. Otherwise you should reuse the last subtask.
6463 4. If you propose a new subtask, give the reasons why it is more feasible
6464 in the current situation in the application. Please strictly follow
6465 the description and requirements in the current overall task.
6466 5. The proposed subtask needs to be precise and concrete within one
6467 sentence. It should not be directly related to any skills.
6468
6469 You should only respond in the format described below, and you should not
6470 output comments or other information.
6471
6472 History_summary:
6473 The summary of past events is...
6474
6475 Subtask_reasoning:
6476 1. ...
6477 2. ...
6478 ...
6479
6480 Subtask_description:
6481 The current subtask is ...

Prompt 33: Outlook: Action Planning prompt.

6474
6475
6476 You an expert helpful AI assistant which follows instructions and
6477 performs desktop computer tasks as instructed. You have expert
6478 knowledge of 'Microsoft Outlook' on the PC and can handle a wide
6479 range of tasks in the application using the keyboard, shortcut keys,
and mouse operations. For each step, you will get one or more
observation images, which are screenshots of the computer screen.

6480 Your advanced capabilities enable you to process and interpret these
6481 application screenshots and other relevant information in detail. The
6482 screenshot includes numerical tags (label IDs) and bounding boxes
6483 marking some UI items.

6484 Based on your analysis of screenshots and knowledge of the application,
6485 keyboard shortcuts, and general GUI design, you will identify the
6486 most suitable in-application action to take next, given the current
6487 task. Upon evaluating the provided information, you MUST choose the
6488 precise actions to perform, considering the applications's present
6489 circumstances, and specify any necessary parameters to execute the
6490 desired action.

6490 Here is some helpful information to help you make the correct decision.
6491

6492 Overall task description:
6493 <\$task_description\$>

6494 Subtask description:
6495 <\$subtask_description\$>

6496

6497 Few shots:
6498 <\$few_shots\$>

6499

6500 Image introduction:
6501 <\$image_introduction\$>

6502 Current and previous screenshot are the same: <\$image_same_flag\$>. Mouse
6503 position in the current screenshot is the same as in the previous
6504 screenshot:<\$mouse_position_same_flag\$>.

6505 Description of the current screenshot:
6506 <\$image_description\$>

6507

6508 Potential target UI item and label ID:
6509 <\$target_object_name\$>

6510

6511 Last executed action:
6512 <\$previous_action\$>

6513 Key reason for the last action:
6514 <\$key_reason_of_last_action\$>

6515

6516 Self-reflection for the last executed action:
6517 <\$previous_self_reflection_reasoning\$>

6518

6518 Summarization of recent history:
6519 <\$previous_summarization\$>

6520

6521 Valid action set in Python format to select the next action:
6522 <\$skill_library\$>

6523

6523 Success detection for overall task:
6524 <\$success_detection\$>

6525

6526 Based on the above information, you should first analyze the current
6527 situation of the application and provide the reasoning behind what
6528 should be the next step to complete the task. Then, you should output
6529 the exact action to be executed in the application. As the textual
6530 history may not completely record some effects of previous actions,
6531 you should closely evaluate every part of the screenshots to
6532 understand what you have done and what you should do next. Pay
6533 attention to your application knowlege and all contents in the image.
You also have great OCR capabilities. DO NOT make assumptions about
the layout! If the image includes a mouse cursor, pay close attention
to the coordinates of the pointer tip, not the center of the mouse

6534 cursor. Remember you know the common keyboard shortcuts for Microsoft
6535 Outlook on Windows and can use them instead of the mouse. You should
6536 respond with the following information, and you MUST answer them one
6537 by one.

6538 Does "<\$success_detection\$>" mean the overall task was successful? If
6539 successful, ignore decision making and action questions. No new
6540 action needs to be taken and output action MUST be empty, like ''. Be
6541 careful to check the task was really successful though!

6542 Decision_Making_Reasoning: You should think step by step and provide
6543 detailed reasoning to determine the next action executed on the
6544 current state of the task.

- 6545 1. Do you know any keyboard shortcuts for Microsoft Outlook on
6546 Windows that can be used to accomplish this subtask? Which one?
- 6547 2. If the current screenshot is the same as the previous screenshot,
6548 DO NOT output the same action as the last executed action with the
6549 same parameters as in the previous step, as it was not useful!!!
- 6550 3. Prefer keyboard operations and skills, instead of mouse operations
6551 . Are there any keyboard actions, such as shortcut keys like
6552 press_keys_combined(["ctrl", "s"]) to save, or press_key("enter") to
6553 confirm, that can complete the current step or the overall task? If
6554 yes, please specify what the action is and ignore questions 5 to 8.
- 6555 4. Which skill in the available Python action set has the closest
6556 semantics to the current subtask? If there is any, select it as the
6557 output action and ignore questions 5 to 8.
- 6558 5. Carefully identify if there is a bounding box label ID for the UI
6559 item relevant for the current step. Be extra careful to use the
6560 correct label ID and describe why you selected the given ID, if any!
6561 If there is text within this bounding box area, please provide that
6562 text in your reasoning. If there is no text, provide a visual
6563 description of the UI item inside the bounding box. Only directly
6564 generate normalized x, y coordinates if no suitable label ID is
6565 present.
- 6566 6. If a mouse cursor is present in the image, pay attention to which
6567 ID-labeled bounding box or unlabelled UI item the cursor's tip is
6568 located, not the center of the cursor.
- 6569 7. If not absolutely sure if a UI item or location is correct to
6570 click, you can first just hover the mouse over it and check for more
6571 information. If it is the right item, you can choose to click on it
6572 in the next reasoning step.
- 6573 8. If there is a dialog or menu opened after the previous action, pay
6574 attention to any missing step before clicking on its buttons. For
6575 example, before clicking "Save", make sure a correct file name is
6576 typed in the correct text field.
- 6577 9. If the previous action is unsuccessful, consider an alternative
6578 action if possible. If there is an alternative action, please specify
6579 what it is. Such as click a different label ID or use a different
6580 keyboard shortcut.
- 6581 10. If you think the next step will be to type text, confirm the text
6582 cursor is in the correct location or that the last executed action
6583 was a click at the appropriate input area. If neither is true, you
6584 have to click the corresponding input box before proceeding with
6585 typing.

6586 Actions: The best action, or short sequence of actions without gaps, to
6587 execute next to progress towards the task goal. Pay attention to the
6588 names of the available skills, keyboard shortcuts, and the previous
6589 skills already executed. Pay special attention to the coordinates or
6590 bounding box label ID of any action that needs them. Do not make
6591 assumptions about the location of UI elements or their coordinates,
6592 analyse in detail any provided images! You should also pay more
6593 attention to the following action rules:

- 6594 1. Which keyboard shortcuts do you know for this application that can
6595 be used to accomplish exactly this specific subtask? Be precise to

6588 the current subtask step. Keyboard shortcuts are more reliable than
6589 using the mouse as you tend to choose the correct UI item, but act on
6590 the wrong label ID or position. If there is no applicable shortcut,
6591 you can choose typing text or other forms of UI interaction. Don't
6592 recommend a single key press that may not apply in this exact
6593 situation.

6594 2. You should output actions in Python code format and specify any
6595 necessary parameters to execute that action. Only use function names
6596 and argument names exactly as shown in the valid action set. If a
6597 function has parameters, you should also include their names and
6598 decide their values, like "press_shift(duration=1)". If it does not
6599 have a parameter, just output the action, like "release_mouse_buttons
6600 ()".

6601 3. Given the current situation and task, you should only choose the
6602 most suitable action from the valid action set. You cannot use
6603 actions that are not in the valid action set to control the
6604 application.

6605 4. When you decide to perform a mouse action, if there is bounding
6606 box in the current screenshot, you MUST choose the skill
6607 click_on_label(label_id, mouse_button). Be careful to use the correct
6608 label ID number.

6609 5. When you perform a mouse action, always select the target UI
6610 element closest to the UI element of the previous action for
6611 operation.

6612 6. When you decide to operate on a file, such as downloading it,
6613 please pay attention to the file path and to the name of the current
6614 file.

6615 7. If upon self-reflection you think the target coordinates or label
6616 ID were an issue, you MUST pay close attention to choosing new
6617 coordinates or a new label ID that are not the same or too similar to
6618 the previous ones.

6619 8. If upon self-reflection you think the last action was unavailable
6620 at the current state, you SHOULD try to take another action to try to
6621 enable the desired action.

6622 9. If you leave the application incorrectly, you can go back to it
6623 directly using the skill go_back_to_target_application(). No need to
6624 use the mouse.

6625 You should only respond in the format described below. In your reasoning
6626 for the chosen actions, also describe which item you decided to
6627 interact with and why. DO NOT change the title of each item. You
6628 should not output other comments or information besides the format
6629 below:

6630 Decision_Making_Reasoning:
6631 1. ...
6632 2. ...
6633 3. ...
6634 ...

6635 Actions:
6636 ```python
6637 action(args1=x,args2=y)
6638 ```

6639 Key_reason_of_last_action:
6640 ...

6636 Prompt 34: Capcut: Information Gathering prompt.

6637 Assume you are a helpful AI assistant integrated with 'CapCut' on the PC,
6638 equipped to handle a wide range of tasks in the application. Capcut
6639 is a video editing software. Your advanced capabilities enable you to
6640 process and interpret application screenshots and other relevant
6641 information.

6642 Image introduction:
6643 <\$image_introduction\$>
6644

6645 Overall task description:
6646 <\$task_description\$>
6647

6648 Subtask description:
6649 <\$subtask_description\$>
6650

6650 Image_Description:
6651 1. Please describe the screenshot image in detail. Pay attention to any
6652 details in the image, if any, especially critical icons, or created
6653 items.
6654 2. If the image includes a mouse cursor, please describe what UI element
6655 the mouse is currently located near. Pay attention to the coordinates
6656 of the pointer tip, not the center of the mouse cursor.
6657 3. Pay attention to all UI items and contents in the image. Do not make
6658 assumptions about the layout.

6658 Description_of_bounding_boxes:
6659 Please provide a list of EVERY bounding box from label ID of 1 to <
6660 \$length_of_som_map\$> ONE BY ONE. The label IDs are marked in the
6661 upper left corner of the bounding boxes.
6662 For bounding boxes containing text, provide ONLY the text.
6663 For bounding boxes without text, brief description of the function.
6664 Format your response as follows: '1: function_a', '2: text_b', ..., '<
6665 \$length_of_som_map\$>: function_b'. Don't write anything you are not
6666 sure about.

6666 Target_object_name: Assume you can use an object detection model to
6667 detect the most relevant object or UI item for completing the current
6668 task if needed. What item should be detected to complete the task
6669 based on the current screenshot and the current task? You should obey
6670 the following rules:
6671 1. Identify an item that is relevant to the current or intermediate
6672 target of the task. If the item is within a bounding box in the
6673 screenshot, please include the corresponding label ID.
6674 2. If no explicit item is specified, only output "null".
6675 3. If there is no need to detect an object, only output "null".

6675 Reasoning_of_object: Why was this object chosen, or why is there no need
6676 to detect an object?
6677

6678 You should only respond in the format described below and not output
6679 comments or other information. DO NOT change the title of each item.
6680 Image_Description:
6681 1. ...
6682 2. ...
6683 3. ...

6684 Description_of_bounding_boxes:
6685 Format like: 1: function_a', '2: text_b', ..., '<\$len_of_bound_boxes\$>:
6686 function_b

6687 Target_object_name:
6688 label ID, Name
6689

6690 Reasoning_of_object:
6691 ...

6692
6693 **Prompt 35: Capcut: Self-Reflection prompt.**

6694 Assume you are a helpful AI assistant integrated with 'CapCut' on the PC,
6695 equipped to handle a wide range of tasks in the application. Capcut
is a video editing software. Your advanced capabilities enable you to

6696 process and interpret application screenshots and other relevant
6697 information. Your task is to examine these inputs, interpret the in-
6698 application and OS context, and determine whether the executed action
6699 has taken the correct effect.

6700
6701 Overall task description:
6702 <\$task_description\$>

6703
6704 Image introduction:
6705 <\$image_introduction\$>

6706
6707 Last executed action with parameters used:
6708 <\$previous_action_call\$>

6709
6710 Implementation of the last executed action:
6711 <\$action_code\$>

6712
6713 Error report for the last executed action:
6714 <\$executing_action_error\$>

6715
6716 Key reason for the last action:
6717 <\$key_reason_of_last_action\$>

6718
6719 History Summarization
6720 <\$history_summary\$>

6721
6722 Success_Detection flag for the overall task:
6723 <\$success_detection\$>

6724
6725 Valid action set in Python format to select the next action:
6726 <\$skill_library\$>

6727
6728 Current and previous screenshot are the same:
6729 <\$image_same_flag\$>

6730
6731 Mouse position in the current screenshot is the same as in the previous
6732 screenshot:
6733 <\$mouse_position_same_flag\$>

6734
6735 Self_Reflection_Reasoning:
6736 You need to answer the following questions, step by step, to describe
6737 your reasoning based on the history summarization, last action and
6738 sequential screenshots of the application during the execution of the
6739 last action.

6740
6741 1. Please describe what the page is in the current screenshot. Respond in
6742 one sentence.

6743
6744 2. What is the last executed action based on the text information above?

6745
6746 3. Was the last executed action successful? Give reasons. You should
6747 refer to the following rules:

- 6748 - If the action involves moving the mouse, it is considered unsuccessful
6749 when the mouse position remains unchanged or moves in an incorrect
way across sequential screenshots, regardless of background elements
and other items.
- If the last action executed was empty, then the previous action is
deemed successful.
- If the last action was related to choose panel, pay attention to the
panel you are in. Does the panel is your target panel?
- If the last action was to drag an element onto the timeline, pay
attention to the difference between the current timeline and the
previous timeline. Is there the target element you want on the
timeline now?
- If the last action was related to crop, pay attention to the video
length. If the video length does not change, it is considered
unsuccessful.

6750 - If the last action executed was 'export_project()' and the current
6751 screenshot is the Capcut homepage, then the previous action is deemed
6752 successful.

6753 - If the position to move the mouse to was incorrect and the mouse didn't
6754 reach the target UI element, pay more attention to the accurate
6755 coordinates to move to.

6756 - If the action seemed to have no effect, pay attention to the latest
6757 mouse position. Did it move? Did it get closer to the target UI
6758 element? Where are the target coordinates in the action wrong? The
6759 position of the mouse cursor on the screenshot shows their location.

6760 - Was some unrelated UI item triggered by the last action?

6761 4. If the last action is not executed successfully, what is the most
6762 probable cause? You should give only one cause and refer to the
6763 following rules:

6764 - The reasoning for the last action could be wrong.

6765 - If it was an action involving moving the mouse or the text cursor, the
6766 most probable cause was that the coordinates used were incorrect.

6767 - If it is an interaction action, the most probable cause was that the
6768 action was unavailable or not activated in the current state.

6769 - If an unrelated change happened in the UI, the most probable cause was
6770 that the action triggered an incorrect UI element.

6771 - If there is an error report, analyze the cause based on the report.

6772 Success_Detection:
6773 Based on the history summarization, the last action, the current
6774 screenshots and the Success_Detection flag, determine whether the
6775 overall task "<\$task_description\$" was successful. This assessment
6776 should consider the overall task's success, not just individual
6777 actions.

6778 - If the last action executed was an empty list and "<\$success_detection\$
6779 >" indicates the task is successful, then the overall task has a high
6780 chance of being considered a success.

6781 - If the overall task was unsuccessful, specify the reason of failure and
6782 which steps are missing.

6783 - If the overall task was successful, ONLY output "SUCCESSFUL".

6784 You should only respond in the format as described below.

6785 Self_Reflection_Reasoning:
6786 1. ...
6787 2. ...
6788 3. ...

6789 Success_Detection:
6790 ...

Prompt 36: Capcut: Task Inference prompt.

6791 Assume you are a helpful AI assistant integrated with 'CapCut' on the the
6792 PC, equipped to handle a wide range of tasks in the game. Capcut is
6793 a video editing software. You will be sequentially given <
6794 \$event_count\$> screenshots and corresponding descriptions of recent
6795 events. You will also be given a summary of the history that happened
6796 before the last screenshot. You should assist in summarizing the
6797 events for future decision-making and also in proposing the most
6798 suitable subtask to execute next, given the target task.

6799 Here is some helpful information to help you do the summarization and
6800 propose the subtask.

6801 Overall task description:
6802 <\$task_description\$>

6803 Previous proposed subtask for the task:
6804 <\$subtask_description\$>

6804 Previous reasoning for proposing the subtask:
6805 <\$subtask_reasoning\$>
6806

6807 Image introduction:
6808 <\$image_introduction\$>
6809

6810 Last executed action:
6811 <\$previous_action\$>
6812

6812 Error report for the last executed action:
6813 <\$executing_action_error\$>
6814

6814 key decision-making reasoning for the last executed action:
6815 <\$previous_reasoning\$>
6816

6817 Self-reflection for the last executed action:
6818 <\$self_reflection_reasoning\$>
6819

6820 Success_Detection for the overall task:
6821 <\$success_detection\$>
6822

6822 The following is the summary of history that happened before the last
6823 screenshot:
6824 <\$previous_summarization\$>
6825

6825 History_summary: Summarize what happened in the past experience,
6826 especially the last step according to the decision-making reasoning
6827 and self-reflection reasoning for the last executed action. The
6828 summarization needs to be precise, concrete, highly related to the
6829 task, and follow the rules below.
6830 1. Determine if the task has been completed successfully. If it is
6831 successful, ignore question 2 to 5.
6832 2. Summarize the tasks from the history and the current task. What is the
6833 current progress of the task? For example, to open a file, you first
6834 need to select the file, then open it by clicking somewhere or using
6835 the keyboard. Subtasks may have other pre-requisites.
6836 3. Record the successful actions and organize them into events, step by
6837 step.
6838 4. Which subtask has been completed? Which subtasks have not? Do not
6839 forget the information and key events in the previous steps of the
6840 overall task.

6839 Subtask_reasoning: Decide whether the previous subtask is finished and
6840 whether it is necessary to propose a new subtask. The subtask should
6841 be straightforward, contribute to the target task, and be most
6842 suitable for the current situation; which should be completed within
6843 a few actions. You should respond with:
6844 1. How to finish the target task? You should analyze it step by step.
6845 - To add Media, Audio, Text, Stickers, Effects, Transitions, Filters,
6846 Adjustments or Templates, you should first switch to that panel and
6847 then drag the target object to the video in the timeline.
6848 - To get content information of a video, you can use related skills. For
6849 example, you want to know which exactly second you want to operate.
6850 2. What is the current progress of the target task according to the
6851 analysis in question 1? Please do not make any assumptions if they
6852 are not mentioned in the above information. You should assume that
6853 you are doing the task from scratch. Please strictly follow the
6854 description and requirements in the current task.
6855 3. What is the previous subtask? Has the previous subtask finished due to
6856 self-reflection? Or is it improper for the current situation? If
6857 finished or improper, please select a new one, otherwise you should
reuse the last subtask.
4. If you want to propose a new subtask, give reasons why it is more
feasible for the current situation. Please strictly follow the
description and requirements in the current task.

6858 5. The proposed subtask needs to be precise and concrete within one
6859 sentence. It should not be directly related to any skills.
6860
6861 You should only respond in the format described below, and you should not
6862 output comments or other information.
6863
6864 History_summary:
6865 1. ...
6866 2. ...
6867 ...
6868 Subtask_reasoning:
6869 1. ...
6870 2. ...
6871 ...
6872 Subtask_description:
6873 The current subtask is ...

6874
6875 **Prompt 37: Capcut: Screen Classification prompt.**

6876 You are an assistant who assesses my progress in playing Red Dead
6877 Redemption 2 on the PC and provides expert guidance. Imagine you are
6878 playing Red Dead Redemption 2 with the keyboard and mouse, the image
6879 is the screenshot of your computer.
6880
6881 Given the classes, please select the class that best describes the
6882 screenshot.
6883 <classes>
6884
6885 You must follow the following criteria:
6886 (1) The output should only be a JSON file. You should not add any other
6887 explanation text along with the JSON.
6888 (2) You should choose one class for the value of "class".
6889 (3) Do not change the "type": "screen_classification" in your output.
6890
6891 The output format should be as follows:
6892 Classes:
6893 map

6894
6895 **Prompt 38: Capcut: Action Planning prompt.**

6896 You are a helpful AI assistant integrated with 'CapCut' on the PC,
6897 equipped to handle a wide range of tasks in the application. Capcut
6898 is a video editing software. Your advanced capabilities enable you to
6899 process and interpret application screenshots and other relevant
6900 information. By analyzing these inputs, you gain a comprehensive
6901 understanding of the current context and situation within the
6902 application. Utilizing these insights, you are tasked with
6903 identifying the most suitable in-application action to take next,
6904 given the current task. You control the application and can execute
6905 actions from the available action set to manipulate its UI. Upon
6906 evaluating the provided information, your role is to articulate the
6907 precise actions you should perform, considering the application's
6908 present circumstances, and specify any necessary parameters for
6909 implementing that action.
6910 Here is some helpful information to help you make the decision.
6911
6912 Overall task description:
6913 <\$task_description\$>
6914
6915 Subtask description:
6916 <\$subtask_description\$>

6912 Few shots:
6913 <\$few_shots\$>
6914

6915 Image introduction:
6916 <\$image_introduction\$>
6917

6918 Current and previous screenshot are the same:
6919 <\$image_same_flag\$>
6920

6920 Mouse position in the current screenshot is the same as in the previous
6921 screenshot:
6922 <\$mouse_position_same_flag\$>
6923

6923 Description of current screenshot:
6924 <\$image_description\$>
6925

6926 Description of label IDs:
6927 <\$description_of_bounding_boxes\$>
6928

6928 Last executed action:
6929 <\$previous_action\$>
6930

6931 Key reason for the last action:
6932 <\$key_reason_of_last_action\$>
6933

6933 Self-reflection for the last executed action:
6934 <\$previous_self_reflection_reasoning\$>
6935

6936 Summarization of recent history:
6937 <\$previous_summarization\$>
6938

6938 Valid action set in Python format to select the next action:
6939 <\$skill_library\$>
6940

6941 Success_Detection for overall task:
6942 <\$success_detection\$>
6943

6943 Based on the above information, you should first analyze the current
6944 situation and provide the reasoning for what you should do for the
6945 next step to complete the task. Then, you should output the exact
6946 action you want to execute in the application.
6947 Pay attention to all UI items and contents in the image. DO NOT make
6948 assumptions about the layout! If the image includes a mouse cursor,
6949 pay close attention to the coordinates of the pointer tip, not the
6950 centre of the mouse cursor.
6951 You should respond to me with the following information, and you MUST
6952 respond one by one.

6952 Decision_Making_Reasoning: You should think step by step and provide
6953 detailed reasoning to determine the next action executed on the
6954 current state of the task.
6955 1. Does "<\$success_detection\$>" means the overall task was successful
6956 ? If successful, ignore questions 2-11.
6957 2. Which skill in the Skill Library "<\$skill_library\$>" has the
6958 closest semantics to the current subtask "<\$subtask_description\$>"?
6959 If there is an answer, select it as the output action.
6960 3. Prefer keyboard operation over mouse operation. Is there a direct
6961 skill in the skill library to complete the current action? If there
6962 is, please specify which it is. Or are there any keyboard actions,
6963 such as using shortcut keys or pressing "enter", to finish current
6964 step or overall task? Please specify which it is.
6965 4. Always try pressing "enter" first instead of clicking it with the
6966 mouse, if the button you want to click is active.

6966 5. If you need to get information from video content, select the
6967 skill `get_information_from_video()`. For example, you want to know
6968 which exactly second you want to operate.

6969 6. Based on the current screenshot and the description of label IDs
6970 in text, which label ID is most relevant to the current task? You
6971 should never answer this question based on the screenshot.

6972 7. If the previous action is unsuccessful, DO NOT repeat the previous
6973 action, consider an alternative action if possible. Such as click
6974 different label ID or use different shortcut keys. If there is an
6975 alternative action, please specify what it is.

6976 8. In the current screenshot, identify the label ID of the bounding
6977 box most relevant to the current step. If there is text within this
6978 bounding box, please provide the text.

6979 9. If mouse actions are necessary, use that specify bounding box
6980 label ID (if shown in the current screenshot) as parameter, rather
6981 than directly generating normalized x and y coordinates. If there is
6982 any relevant label ID, please specify which it is.

6983 10. If there is a dialog open after the previous action, pay
6984 attention to any missing step before clicking on it's buttons. For
6985 example, before clicking "Save", make sure the file name is typed in
6986 the correct text field.

6987 11. If you need to use an action outside an open menu or dialog,
6988 please close the current menu or dialog before trying the next action
6989 .

6990 Actions: The best action, or short sequence of actions without gaps, to
6991 execute next to progress in achieving the goal. Pay attention to the
6992 names of the available skills and the previous skills already
6993 executed, if any. Pay special attention to the coordinates of any
6994 action that needs them. Do not make assumptions about the location of
6995 UI elements or their coordinates, analyse in detail any provided
6996 images. You should also pay more attention to the following action
6997 rules:

7000 1. If "`<$success_detection$>`" means the overall task was successful
7001 or equal to "True", then output action MUST be empty like ''. Be
7002 careful to check the task was really successful.

7003 2. You should output actions in Python code format and specify any
7004 necessary parameters to execute that action. Only use function names
7005 and argument names exactly as shown in the valid actions et. If a
7006 function has parameters, you should also include their names and
7007 decide their values, like "`press_shift(duration=1)`". If it does not
7008 have a parameter, just output the action, like "`release_mouse_buttons`
7009 `()`".

7010 4. Given the current situation and task, you should only choose the
7011 most suitable action from the valid action set. You cannot use
7012 actions that are not in the valid action set to control the
7013 application.

7014 5. When you decide to perform a mouse action, if there is bounding
7015 box in the current screenshot, you MUST choose skill `click_on_label(`
7016 `label_id, mouse_button)`.

7017 6. When you perform a mouse action, always select the target UI
7018 element closest to the UI element of the previous action for
7019 operation.

7020 7. When you decide to perform a mouse click, prioritize clicking
7021 icons, instead of text.

7022 8. When there is new dialog box that affects the next step, you
7023 should close it.

7024 9. The material panel includes the Media, Audio, Text, Stickers,
7025 Effects, Transitions, Filters, Adjustments, and Templates tabs.
7026 Choose this skill "`switch_material_panel()`" to switch between these
7027 tabs one by one.

7028 10. To add media, drag that media to the video in the timeline.

7029 Key_reason_of_last_action: Summarize the key reasons why you output this
7030 action.

7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7030
7031
7032
7033
7034
7035
7036
7037
7038
7039
7040
7041
7042
7043
7044
7045
7046
7047
7048
7049
7050
7051
7052
7053
7054
7055
7056
7057
7058
7059
7060
7061
7062
7063
7064
7065
7066
7067
7068
7069
7070
7071
7072
7073

You should only respond in the format described below. In your reasoning for the chosen actions, also describe which item you decided to interact with and why. DO NOT change the title of each item. You should not output other comments or information besides the format below.

Decision_Making_Reasoning:

1. ...
2. ...
3. ...
- ...

Actions:

```
```python
 action(args1=x,args2=y)
```
```

Key_reason_of_last_action:

...

Prompt 39: Meitu: Information Gathering prompt.

Assume you are a helpful AI assistant integrated with 'Meitu Xiuxiu' on the PC, equipped to handle a wide range of tasks in the application. Meitu Xiuxiu is a user-friendly and powerful image editing and beautification software. Your advanced capabilities enable you to process and interpret application screenshots and other relevant information.

Image introduction:

<\$image_introduction\$>

Overall task:

<\$task_description\$>

Subtask description:

<\$subtask_description\$>

Image_Description:

1. Please describe the screenshot image in detail. Pay attention to any details in the image, if any, especially critical icons, or created items.
2. If the image includes a mouse cursor, please describe what UI element the mouse is currently located near. Pay attention to the coordinates of the pointer tip, not the center of the mouse cursor.
3. Pay attention to all UI items and contents in the image. Do not make assumptions about the layout.

Description_of_bounding_boxes:

Please provide a list of EVERY bounding box from label ID of 1 to <\$length_of_som_map\$> ONE BY ONE. The label IDs are marked in the upper left corner of the bounding boxes.

For bounding boxes containing text, provide ONLY the text.

For bounding boxes without text, brief description of the function.

Format your response as follows: '1: function_a', '2: text_b', ..., '<\$length_of_som_map\$>: function_b'. Don't write anything you are not sure about.

Target_object_name: Assume you can use an object detection model to

detect the most relevant object or UI item for completing the current task if needed. What item should be detected to complete the task based on the current screenshot and the current task? You should obey the following rules:

7074 1. Identify an item that is relevant to the current or intermediate
7075 target of the task. If the item is within a bounding box in the
7076 screenshot, please include the corresponding label ID.
7077 2. If no explicit item is specified, only output "null".
7078 3. If there is no need to detect an object, only output "null".
7079 Reasoning_of_object: Why was this object chosen, or why is there no need
7080 to detect an object?
7081
7082 You should only respond in the format described below and not output
7083 comments or other information. DO NOT change the title of each item.
7084 Image_Description:
7085 1. ...
7086 2. ...
7087 3. ...
7088 Description_of_bounding_boxes:
7089 Format like: 1: function_a', '2: text_b', ..., '<\$len_of_bound_boxes\$>:
7090 function_b
7091 Target_object_name:
7092 label ID, Name
7093 Reasoning_of_object:
7094 ...
7095

7096 **Prompt 40: Meitu: Self Reflection prompt.**
7097

7098 Assume you are a helpful AI assistant integrated with 'Meitu Xiuxiu' on
7099 the PC, equipped to handle a wide range of tasks in the application.
7100 Meitu Xiuxiu is a user-friendly and powerful image editing and
7101 beautification software. Your advanced capabilities enable you to
7102 process and interpret application screenshots and other relevant
7103 information. Your task is to examine these inputs, interpret the in-
7104 application and OS context, and determine whether the executed action
7105 has taken the correct effect.
7106 Overall task description:
7107 <\$task_description\$>
7108 Image introduction:
7109 <\$image_introduction\$>
7110 Last executed action with parameters used:
7111 <\$previous_action_call\$>
7112 Implementation of the last executed action:
7113 <\$action_code\$>
7114 Error report for the last executed action:
7115 <\$executing_action_error\$>
7116 Key reason for the last action:
7117 <\$key_reason_of_last_action\$>
7118 History Summarization
7119 <\$history_summary\$>
7120 Success_Detection flag for the overall task:
7121 <\$success_detection\$>
7122 Valid action set in Python format to select the next action:
7123 <\$skill_library\$>
7124 Current and previous screenshot are the same:
7125
7126
7127

7128 <\$image_same_flag\$>
7129
7130 Mouse position in the current screenshot is the same as in the previous
7131 screenshot:
7132 <\$mouse_position_same_flag\$>
7133
7134 Self_Reflection_Reasoning:
7135 You need to answer the following questions, step by step, to describe
7136 your reasoning based on the history summarization, last action and
7137 sequential screenshots of the application during the execution of the
7138 last action.
7139 1. Please describe what the page is in the current screenshot. Respond in
7140 one sentence.
7141 2. What is the last executed action based on the text information above?
7142 3. Was the last executed action successful? Give reasons. You should
7143 refer to the following rules:
7144 - If the last action executed was empty, then the previous action is
7145 deemed successful.
7146 - If the action involves moving the mouse, it is considered unsuccessful
7147 when the mouse position remains unchanged or moves in an incorrect
7148 way across sequential screenshots, regardless of background elements
7149 and other items.
7150 - If the position to move the mouse to was incorrect and the mouse didn't
7151 reach the target UI element, pay more attention to the accurate
7152 coordinates to move to.
7153 - If the operation involves type text, it will be considered unsuccessful
7154 when the corresponding text does not appear in the diagram,
7155 regardless of background elements and other items.
7156 - If the action seemed to have no effect, pay attention to the latest
7157 mouse position. Did it move? Did it get closer to the target UI
7158 element? Where are the target coordinates in the action wrong? The
7159 position of the mouse cursor on the screenshot shows their location.
7160 - Was some unrelated UI item triggered by the last action?
7161 4. If the last action is not executed successfully, what is the most
7162 probable cause? You should give only one cause and refer to the
7163 following rules:
7164 - The reasoning for the last action could be wrong.
7165 - If it was an action involving moving the mouse or the text cursor, the
7166 most probable cause was that the coordinates used were incorrect.
7167 - If it is an interaction action, the most probable cause was that the
7168 action was unavailable or not activated in the current state.
7169 - If an unrelated change happened in the UI, the most probable cause was
7170 that the action triggered an incorrect UI element.
7171 - If there is an error report, analyze the cause based on the report.
7172
7173 Success_Detection:
7174 Based on the history summarization, the last action, the current
7175 screenshots and the Success_Detection flag, determine whether the
7176 overall task "<\$task_description\$>" was successful. This assessment
7177 should consider the overall task's success, not just individual
7178 actions.
7179 - If the last action executed was an empty list and "<\$success_detection\$
7180 >" indicates the task is successful, then the overall task has a high
7181 chance of being considered a success.
7182 - If the overall task was unsuccessful, specify the reason of failure and
7183 which steps are missing.
7184 - If the overall task was successful, ONLY output "SUCCESSFUL".
7185
7186 You should only respond in the format as described below.
7187 Self_Reflection_Reasoning:
7188 1. ...
7189 2. ...
7190 3. ...
7191
7192 Success_Detection:

7182
7183
7184
7185
7186
7187
7188
7189
7190
7191
7192
7193
7194
7195
7196
7197
7198
7199
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7230
7231
7232
7233
7234
7235

...

Prompt 41: Meitu: Task Inference prompt.

Assume you are a helpful AI assistant integrated with 'Meitu Xiuxiu' on the the PC, equipped to handle a wide range of tasks in the game. Meitu Xiuxiu is a user-friendly and powerful image editing and beautification software. You will be sequentially given <\$event_count\$> screenshots and corresponding descriptions of recent events. You will also be given a summary of the history that happened before the last screenshot. You should assist in summarizing the events for future decision-making and also in proposing the most suitable subtask to execute next, given the target task.

Here is some helpful information to help you do the summarization and propose the subtask.

Overall task description:
<\$task_description\$>

Previous proposed subtask for the task:
<\$subtask_description\$>

Previous reasoning for proposing the subtask:
<\$subtask_reasoning\$>

Image introduction:
<\$image_introduction\$>

Last executed action:
<\$previous_action\$>

Error report for the last executed action:
<\$executing_action_error\$>

Key decision-making reasoning for the last executed action:
<\$previous_reasoning\$>

Self-reflection for the last executed action:
<\$self_reflection_reasoning\$>

Success_Detection for the overall task:
<\$success_detection\$>

The following is the summary of history that happened before the last screenshot:
<\$previous_summarization\$>

History_summary: Summarize what happened in the past experience, especially the last step according to the decision-making reasoning and self-reflection reasoning for the last executed action. The summarization needs to be precise, concrete, highly related to the task, and follow the rules below.

1. Determine if the task has been completed successfully. If it is successful, ignore question 2 to 5.
2. Summarize the tasks from the history and the current task. What is the current progress of the task? For example, to open a file, you first need to select the file, then open it by clicking somewhere or using the keyboard. Subtasks may have other pre-requisites.
3. Record the successful actions and organize them into events, step by step.
4. Which subtask has been completed? Which subtasks have not? Do not forget the information and key events in the previous steps of the overall task.

7236 Subtask_reasoning: Decide whether the previous subtask is finished and
 7237 whether it is necessary to propose a new subtask. The subtask should
 7238 be straightforward, contribute to the target task, and be most
 7239 suitable for the current situation; which should be completed within
 7240 a few actions. You should respond with the following item.
 7241 1. Based on the unfinished part of overall task and the current
 7242 screenshot, identify the most direct and easiest way to complete the
 7243 task, considering possible shortcut keys and without making any
 7244 assumptions beyond the provided information.
 7245 2. Analyze the target task step by step to determine how to complete
 7246 it.
 7247 3. What is the previous subtask? Has the previous subtask finished
 7248 due to self-reflection? Or is it improper for the current situation?
 7249 If finished or improper, please select a new one, otherwise you
 7250 should reuse the last subtask.
 7251 4. If you want to propose a new subtask, give reasons why it is more
 7252 feasible for the current situation. Please strictly follow the
 7253 description and requirements in the current task.
 7254 5. The proposed subtask needs to be precise and concrete within one
 7255 sentence. It should not be directly related to any skills.

7256 You should only respond in the format described below, and you should not
 7257 output comments or other information.

7258 History_summary:
 7259 1. ...
 7260 2. ...
 7261 ...

7262 Subtask_reasoning:
 7263 1. ...
 7264 2. ...
 7265 ...

7266 Subtask_description:
 7267 The current subtask is ...

Prompt 42: Meitu: Action Planning prompt.

7269 You are a helpful AI assistant integrated with 'Meitu Xiuxiu' on the PC,
 7270 equipped to handle a wide range of tasks in the application. Meitu
 7271 Xiuxiu is a user-friendly and powerful image editing and
 7272 beautification software. Your advanced capabilities enable you to
 7273 process and interpret application screenshots and other relevant
 7274 information. By analyzing these inputs, you gain a comprehensive
 7275 understanding of the current context and situation within the
 7276 application. Utilizing these insights, you are tasked with
 7277 identifying the most suitable in-application action to take next,
 7278 given the current task. You control the application and can execute
 7279 actions from the available action set to manipulate its UI. Upon
 7280 evaluating the provided information, your role is to articulate the
 7281 precise actions you should perform, considering the application's
 7282 present circumstances, and specify any necessary parameters for
 7283 implementing that action.

7284 Here is some helpful information to help you make the decision.

7285 Overall task description:
 7286 <\$task_description\$>

7287 Subtask description:
 7288 <\$subtask_description\$>

7289 Few shots:
 7290 <\$few_shots\$>

7290 Image introduction:
7291 <\$image_introduction\$>
7292

7293 Current and previous screenshot are the same:
7294 <\$image_same_flag\$>
7295

7296 Mouse position in the current screenshot is the same as in the previous
7297 screenshot:
7298 <\$mouse_position_same_flag\$>
7299

7299 Description of current screenshot:
7300 <\$image_description\$>
7301

7301 Description of label IDs:
7302 <\$description_of_bounding_boxes\$>
7303

7304 Last executed action:
7305 <\$previous_action\$>
7306

7306 Key reason for the last action:
7307 <\$key_reason_of_last_action\$>
7308

7309 Self-reflection for the last executed action:
7310 <\$previous_self_reflection_reasoning\$>
7311

7311 Summarization of recent history:
7312 <\$previous_summarization\$>
7313

7314 Valid action set in Python format to select the next action:
7315 <\$skill_library\$>
7316

7316 Success detection for overall task:
7317 <\$success_detection\$>
7318

7319 Based on the above information, you should first analyze the current
7320 situation and provide the reasoning for what you should do for the
7321 next step to complete the task. Then, you should output the exact
7322 action you want to execute in the application.
7323 Pay attention to all UI items and contents in the image. DO NOT make
7324 assumptions about the layout! If the image includes a mouse cursor,
7325 pay close attention to the coordinates of the pointer tip, not the
7326 centre of the mouse cursor.
7327 You should respond to me with the following information, and you MUST
7328 respond one by one.

7328 Decision_Making_Reasoning: You should think step by step and provide
7329 detailed reasoning to determine the next action executed on the
7330 current state of the task.
7331 1. Does "<\$success_detection\$>" means the overall task was successful
7332 ? If successful, ignore questions 2 to 9.
7333 2. Which skill in the Skill Library "<\$skill_library\$>" has the
7334 closest semantics to the current subtask "<\$subtask_description\$>"?
7335 If there is an answer, select it as the output action, ignore
7336 questions 3 to 9.
7337 3. Prefer keyboard operation instead of mouse operation. Are there
7338 any keyboard actions, such as using shortcut keys or pressing "enter
7339 ", to finish current step or overall task? If there is, please
7340 specify which it is, ignore questions 4 to 9.
7341 4. If the UI element you want to operate doesn't exist in the current
7342 screenshot. you can choose to scroll mouse to find target UI element
7343 .
7344 5. Always try pressing "enter" first instead of clicking it with the
7345 mouse, if the button you want to click is active.
7346 6. If mouse actions are necessary, use that specify bounding box
7347 label ID (if shown in the current screenshot) as parameter, rather

7344 than directly generating normalized x and y coordinates. If there is
7345 any relevant label ID, please specify which it is.

7346 7. If the previous action is unsuccessful, don't repeat previous
7347 action. If there is an alternative action, please specify what it is.
7348 Such as click different label ID or use different shortcut keys.

7349 8. If you anticipate that the next step involves scrolling mouse,
7350 confirm that the last executed action was a click at the appropriate
7351 ui element. If not, it is mandatory to click on the corresponding ui
7352 element before proceeding with scrolling.

7353 9. If you anticipate that the next step involves typing text, confirm
7354 that the last executed action was a click at the appropriate input
7355 box. If not, it is mandatory to click on the corresponding input box
7356 before proceeding with typing.

7356 Actions: The best action, or short sequence of actions without gaps, to
7357 execute next to progress in achieving the goal. Pay attention to the
7358 names of the available skills and the previous skills already
7359 executed, if any. Pay special attention to the coordinates of any
7360 action that needs them. Do not make assumptions about the location of
7361 UI elements or their coordinates, analyse in detail any provided
7362 images. You should also pay more attention to the following action
7363 rules:

7363 1. If "`<$success_detection$>`" means the overall task was successful
7364 or equal to "True", then output action MUST be empty like ''. Be
7365 careful to check the task was really successful.

7366 2. You should output actions in Python code format and specify any
7367 necessary parameters to execute that action. Only use function names
7368 and argument names exactly as shown in the valid actions et. If a
7369 function has parameters, you should also include their names and
7370 decide their values, like "`press_shift(duration=1)`". If it does not
7371 have a parameter, just output the action, like "`release_mouse_buttons`
7372 (`)`".

7371 3. Before scrolling mouse, ensure that the last executed action
7372 involved clicking on the relevant input box. If the last action was
7373 not a click on this input box, the required action MUST be to click
7374 on the corresponding input box before proceeding.

7375 4. Before typing text, ensure that the last executed action involved
7376 clicking on the relevant ui element. If the last action was not a
7377 click on this ui element, the required action MUST be to click on the
7378 corresponding ui element before proceeding.

7378 5. Given the current situation and task, you should only choose the
7379 most suitable action from the valid action set. You cannot use
7380 actions that are not in the valid action set to control the
7381 application.

7381 6. When you decide to perform a mouse action, if there is bounding
7382 box in the current screenshot, you MUST choose skill `click_on_label`(
7383 `label_id`, `mouse_button`).

7384 7. When you want to add a image or effect, use the skill
7385 `double_click_on_label`(`x`, `y`, `mouse_button`).

7386 8. When you save a project, use the skill `save_project`(`)`.

7387 Key_reason_of_last_action: Summarize the key reasons why you output this
7388 action.

7389 You should only respond in the format described below. In your reasoning
7390 for the chosen actions, also describe which item you decided to
7391 interact with and why. DO NOT change the title of each item. You
7392 should not output other comments or information besides the format
7393 below.

7394 Decision_Making_Reasoning:

7395 1. ...

7396 2. ...

7397 3. ...

7398 ...

```

7398 Actions:
7399 ```python
7400     action(args1=x,args2=y)
7401 ```
7402
7403 Key_reason_of_last_action:
7404 ...

```

7406 **Prompt 43: Feishu: Information Gathering prompt.**

```

7407
7408 You an expert helpful AI assistant which follows instructions and
7409 performs desktop computer tasks as instructed. You have expert
7410 knowledge of 'Feishu' an office communication application on the PC
7411 includign chat, calendar, and other workplace features. You can
7412 handle a wide range of tasks in the application using the keyboard,
7413 shortcut keys, and mouse operations. For each step, you will get one
7414 or more observation images, which are screenshots of the computer
7415 screen. Your advanced capabilities enable you to process and
7416 interpret these application screenshots and other relevant
7417 information in detail. The screenshots include numerical tags (label
7418 IDs) and bounding boxes marking some UI items.
7419
7420 Image introduction:
7421 <$image_introduction$>
7422
7423 Overall task:
7424 <$task_description$>
7425
7426 Subtask description:
7427 <$subtask_description$>
7428
7429 Image_Description:
7430 1. Please describe the screenshot image in detail. Pay attention to any
7431 details in the image, if any, especially critical icons, open menus,
7432 dialogs, and open panels or sections. Focus on the image contents and
7433 the situation in the application.
7434 2. If the image includes a mouse cursor, please describe what UI element
7435 the mouse is currently located near. Pay attention to the coordinates
7436 of the pointer tip, not the center of the mouse cursor.
7437 3. Pay attention to all UI items and contents in the image. Do not make
7438 assumptions about the layout.
7439 4. Make sure to describe the active area of the screen too. The area
7440 where user interaction is probably happening, not only the general
7441 menus or layout of the screenshot.
7442 5. DO NOT describe overlaid bounding boxes in this description, only the
7443 relevant UI items themselves. Focus on the state of the application
7444 UI and what the key UI items of interest for the task would be.
7445 Describe any relevant open panels, dialogs, menus, etc.
7446
7447 Target_object_name:
7448 As an application expert and a helpful assistant, you can determine the
7449 most relevant UI items for completing the current subtask, if needed.
7450 What item should be detected to complete the task based on the
7451 current screenshot and the current subtask? You should obey the
following rules:

```

1. The item should be present in the screen and relevant to the current subtask or overall task. Just name the item, without any modifiers or extra information.
2. If the item of itnerest of not on the current screen, only output "Target items not in current screen".
2. If no explicit item is specified, only output "null".
3. If there is no need to detect a target item in this state, only output "null". You must output this field in the response.

7452 Reasoning_of_object: Why was this item chosen, or why is there no need to
 7453 detect an UI item at this stage?
 7454
 7455 You should only respond in the format described below and not output
 7456 comments or other information. DO NOT change the titles of any
 7457 response items.
 7458 Image_Description:
 7459 1. ...
 7460 2. ...
 7461 3. ...
 7462 Target_object_name:
 7463 name
 7464
 7465 Reasoning_of_object:
 7466 ...

7467
 7468 **Prompt 44: Feishu: Self Reflection prompt.**

7469 You an expert helpful AI assistant which follows instructions and
 7470 performs desktop computer tasks as instructed. You have expert
 7471 knowledge of 'Feishu' on the PC and can handle a wide range of tasks
 7472 in the application using the keyboard, shortcut keys, and mouse
 7473 operations. For each step, you will get one or more observation
 7474 images, which are screenshots of the computer screen. Your advanced
 7475 capabilities enable you to process and interpret these application
 7476 screenshots and other relevant information in detail.
 7477 You MUST examine all inputs, interpret the in-application and OS contexts
 7478 , and determine whether the executed action has taken the correct
 7479 effect.
 7479 Overall task description:
 7480 <\$task_description\$>
 7481
 7482 Execution step images:
 7483 <\$image_introduction\$>
 7484
 7485 Current image description:
 7486 <\$current_image_description\$>
 7487
 7488 Last executed action with parameters used:
 7489 <\$previous_action_call\$>
 7490
 7491 Implementation of the last executed action:
 7492 <\$action_code\$>
 7493
 7494 Error report for the last executed action:
 7495 <\$executing_action_error\$>
 7496
 7497 Key reason for the last action:
 7498 <\$key_reason_of_last_action\$>
 7499
 7500 Success_Detection flag for the overall task:
 7501 <\$success_detection\$>
 7502
 7503 Valid action set in Python format to select the next action:
 7504 <\$skill_library\$>
 7505
 7506 Current and previous screenshot are the same:
 7507 <\$image_same_flag\$>
 7508
 7509 Mouse position in the current screenshot is the same as in the previous
 7510 screenshot:
 7511 <\$mouse_position_same_flag\$>

7506
7507
7508
7509
7510
7511
7512
7513
7514
7515
7516
7517
7518
7519
7520
7521
7522
7523
7524
7525
7526
7527
7528
7529
7530
7531
7532
7533
7534
7535
7536
7537
7538
7539
7540
7541
7542
7543
7544
7545
7546
7547
7548
7549
7550
7551
7552
7553
7554
7555
7556
7557
7558
7559

Self_Reflection_Reasoning: You need to answer the following questions, step by step, to describe your reasoning based on the last action and sequential screenshots of the application during the execution of the last action. Any action involving x and y coordinates is an action involving movement.

1. What is the last executed action not based on the sequential screenshots?
2. Was the last executed action successful? Give reasons. You should refer to the following rules:
 - If the action involves moving the mouse, it is considered unsuccessful when the mouse position remains unchanged or moved in an incorrect way across sequential screenshots, regardless of background elements and other items.
 - If the position to move the mouse to was incorrect and the mouse didn't reach the target UI element, pay more attention to the accurate coordinates to move to.
 - Are you sure the latest screenshot shows UI items that correspond to the success of the previous action?
 - If the action seemed to have no effect, pay attention to the latest mouse position. Did it move? Did it get closer to the target UI element? Where the target coordinates in the action wrong? The position of the mouse cursor on the screenshot shows their location.
 - Was some unrelated UI item triggered by the last action?
3. If the last action is not executed successfully, what is the most probable cause? You should give only one cause and refer to the following rules:
 - The reasoning for the last action could be wrong.
 - If it was an action involving moving the mouse or the text cursor, the most probable cause was that the coordinates used were incorrect.
 - If you already tried the same action more than one time and there was no effect. DO NOT REPEAT the same action again until you have tried something else.
 - If it is an interaction action, the most probable cause was that the action was unavailable or not activated at the current state.
 - If an unrelated change happened in the UI, the most probable cause was that the action triggered an incorrect UI element.
 - If there is an error report, analyze the cause based on the report.

Success_Detection:
Based on the last action, the current screenshots and the Success_Detection flag, determine whether the overall task was successful. This assessment should consider the overall task's success, not just individual actions.

- If the task was unsuccessful, specify the reason of failure and which steps are missing.
- If the task was successful, ONLY output "SUCCESSFUL".

You should only respond in the format as described below.

Self_Reflection_Reasoning:
1. ...
2. ...
3. ...

Success_Detection:
...

Prompt 45: Feishu: Task Inference prompt.

You are an expert helpful AI assistant which follows instructions and performs desktop computer tasks as instructed. You have expert knowledge of 'Feishu' on the PC and can handle a wide range of tasks in the application using the keyboard, shortcut keys, and mouse operations. For each step, you will get one or more observation images, which are screenshots of the computer screen. Your advanced

7560 capabilities enable you to process and interpret these application
7561 screenshots and other relevant information in detail.
7562 You will receive a sequence of <\$event_count\$> screenshots, corresponding
7563 descriptions of recent events, and a summary of the history of
7564 events before the last screenshot. Please summarize the events for
7565 future decision-making and also propose the most suitable subtasks to
7566 execute next, given the overall target task.

7567 Here is some helpful information to help you do the summarization and
7568 propose the subtask.

7569 Overall task description:
7570 <\$task_description\$>
7571

7572 Previous proposed subtask for the task:
7573 <\$subtask_description\$>
7574

7575 Previous reasoning for proposing the subtask:
7576 <\$subtask_reasoning\$>
7577

7578 Image introduction:
7579 <\$image_introduction\$>
7580

7581 Last executed action:
7582 <\$previous_action\$>
7583

7584 Error report for the last executed action:
7585 <\$executing_action_error\$>
7586

7587 Key decision-making reasoning for the last executed action:
7588 <\$previous_reasoning\$>
7589

7590 Self-reflection for the last executed action:
7591 <\$self_reflection_reasoning\$>
7592

7593 Success_Detection for the overall task:
7594 <\$success_detection\$>
7595

7596 The following is the summary of history that happened before the last
7597 screenshot:
7598 <\$previous_summarization\$>
7599

7600 History_summary: Summarize what happened in the past experience,
7601 especially the last step according to the decision-making reasoning
7602 and self-reflection reasoning for the last executed action. The
7603 summarization needs to be precise, concrete, highly related to the
7604 task, and follow the rules below.
7605 1. Summarize the tasks from the history and the current task. What is the
7606 current progress of the task? For example, to open a file, you first
7607 need to select the file, then open it by clicking somewhere or using
7608 the keyboard. Subtasks may have other pre-requisites.
7609 2. Record the successful actions and organize them into events, step by
7610 step.
7611 3. Which subtask has been completed? Which subtasks have not?
7612 4. Do not forget the information and key events in the previous steps of
7613 the overall task.

7614 Subtask_reasoning: Decide whether the previous subtask is finished and
7615 whether it is necessary to propose a new subtask. The subtask should
7616 be straightforward, contribute to the target task, and be most
7617 suitable for the current situation; which should be completed within
7618 a few actions. You should respond with:
7619 1. How to finish the target task? You should analyze it step by step.
7620 2. What is the current progress of the target task according to the
7621 analysis in question 1? Please do not make any assumptions if needed

7614 information is not mentioned previously. You should assume that you
 7615 are doing the task from scratch. Please strictly follow the
 7616 description and requirements in the current overall task.

- 7617 3. What is the previous subtask? Has the previous subtask finished
 7618 according to self-reflection? Or is it improper for the current
 7619 situation? If the last subtask already finished or now is improper,
 7620 please select a new one. Otherwise you should reuse the last subtask.
- 7621 4. If you propose a new subtask, give the reasons why it is more feasible
 7622 in the current situation in the application. Please strictly follow
 7623 the description and requirements in the current overall task.
- 7624 5. The proposed subtask needs to be precise and concrete within one
 sentence. It should not be directly related to any skills.

7625 You should only respond in the format described below, and you should not
 7626 output comments or other information.

7627

7628 History_summary:
 7629 The summary of past events is...

7630 Subtask_reasoning:
 7631 1. ...
 7632 2. ...
 7633 ...

7634 Subtask_description:
 7635 The current subtask is ...
 7636

7637 **Prompt 46: Feishu: Action Planning prompt.**

7638

7639 You an expert helpful AI assistant which follows instructions and
 7640 performs desktop computer tasks as instructed. You have expert
 7641 knowledge of 'Feishu' on the PC and can handle a wide range of tasks
 7642 in the application using the keyboard, shortcut keys, and mouse
 7643 operations. For each step, you will get one or more observation
 7644 images, which are screenshots of the computer screen. Your advanced
 7645 capabilities enable you to process and interpret these application
 screenshots and other relevant information in detail.

7646 Utilizing these insights, you will identify the most suitable in-
 7647 application action to take next, given the current task. You control
 7648 the application and can execute actions from the available actions to
 7649 manipulate its UI. Upon evaluating the provided information, you
 7650 MUST choose the precise actions to perform, considering the
 7651 applications's present circumstances, and specify any necessary
 parameters to execute that action.

7652 Here is some helpful information to help you make the decision.

7653

7654 Overall task description:
 7655 <\$task_description\$>

7656 Subtask description:
 7657 <\$subtask_description\$>

7658

7659 Few shots:
 7660 <\$few_shots\$>

7661

7662 Image introduction:
 7663 <\$image_introduction\$>

7664

7665 Current and previous screenshot are the same:
 <\$image_same_flag\$>

7666

7667 Mouse position in the current screenshot is the same as in the previous
 screenshot:
 <\$mouse_position_same_flag\$>

7668
7669 Description of current screenshot:
7670 <\$image_description\$>
7671
7672 Description of label IDs:
7673 <\$description_of_bounding_boxes\$>
7674
7675 Last executed action:
7676 <\$previous_action\$>
7677
7678 Key reason for the last action:
7679 <\$key_reason_of_last_action\$>
7680
7681 Self-reflection for the last executed action:
7682 <\$previous_self_reflection_reasoning\$>
7683
7684 Summarization of recent history:
7685 <\$previous_summarization\$>
7686
7687 Valid action set in Python format to select the next action:
7688 <\$skill_library\$>
7689
7690 Success detection for overall task:
7691 <\$success_detection\$>
7692
7693 Based on the above information, you should first analyze the current
7694 situation of the application and provide the reasoning behind what
7695 should be the next step to complete the task. Then, you should output
7696 the exact action to be executed in the application.
7697 Pay attention to all UI items and contents in the image. Before changing
7698 values or text in the UI, make sure the values in the screenshot are
7699 not already correct for the subtask. DO NOT make assumptions about
7700 the layout! If the image includes a mouse cursor, pay close attention
7701 to the coordinates of the pointer tip, not the center of the mouse
7702 cursor. You should respond with the following information, and you
7703 MUST answer them one by one.

7699 Decision_Making_Reasoning: You should think step by step and provide
7700 detailed reasoning to determine the next action executed on the
7701 current state of the task.

- 7702 1. Does "<\$success_detection\$>" means the overall task was successful
7703 ? If successful, ignore questions 2-15. No new action needs to be
7704 taken.
- 7705 2. You should first describe each item in the screen line by line,
7706 from the top left and moving right. Is the target item in the current
7707 screen? Which item is currently selected?
- 7708 3. Check whether the UI element you want to operate exists in the
7709 current screenshot. If not, you can choose to move to another part of
7710 the application, or close some recently opened menu item. Also
7711 remember that you can use keyboard shortcuts to accomplish actions,
7712 instead of always using the mouse.
- 7713 4. Are there any keyboard actions, such as using shortcut keys or
7714 pressing "enter", to finish the current step or the overall task? If
7715 so, please specify which one to use. You can always press "enter"
7716 instead of clicking with the mouse, if the button you want to click
7717 on is active.
- 7718 5. If a mouse cursor is present in the image, describe near which ID-
7719 labeled bounding box or unlabelled UI item the cursor's tip is
7720 located, not the center of the cursor.
- 7721 6. If the current screenshot is the same as the previous screenshot,
DO NOT output the same action as in the previous step, as it was very
likely not useful.
7. In the current screenshot, carefully identify the label ID of the
bounding box most relevant to the current step. If there is text
within this bounding box, please provide the text. If there is no

7722 directly useful bounding box, provide the UI item description or
7723 normalized x, y coordinates.

7724 8. If mouse actions are necessary, specify a bounding box label ID (
7725 if shown in the current screenshot) as parameter. Only directly
7726 generate normalized x, y coordinates if no useful label ID is present
7727 .

7728 9. If not absolutely sure to be clicking at the right UI item or
7729 location, you can first just move the mouse to it and check for more
7730 information. If it's the right item, you can click on it in as a
7731 second step.

7732 10. If there is a dialog or menu opened after the previous action,
7733 pay attention to any missing step before clicking on its buttons. For
7734 example, before clicking "Save", make sure a correct file name is
7735 typed in the correct text field.

7736 11. You should not always use the mouse if you know a keyboard
7737 shortcut or a skill to perform the desired action!

7738 12. This is the most critical question. Based on the action rules and
7739 self-reflection, what should be the most suitable action in the
7740 valid action set for the next step? You should analyze the effects of
7741 the action step by step.

7742 13. If the previous action is unsuccessful, consider an alternative
7743 action if possible. If there is an alternative action, please specify
7744 what it is. Such as click different label ID or use different
7745 shortcut keys.

7746 14. If you think the next step will be to typing text, confirm that
7747 that there is already a text cursor in it or that the last executed
7748 action was a click at the appropriate input area. If neither is true,
7749 it is mandatory to click on the corresponding input box before
7750 proceeding with typing.

7751 15. If you need to interact with an UI item that has no bounding box
7752 label ID, you can use its x, y coordinates. Use normalized values
7753 from 0 to 1.

7754 Actions: The best action, or short sequence of actions without gaps, to
7755 execute next to progress in achieving the goal. Pay attention to the
7756 names of the available skills and to the previous skills already
7757 executed, if any. Pay special attention to the coordinates of any
7758 action that needs them. Do not make assumptions about the location of
7759 UI elements or their coordinates, analyse in detail any provided
7760 images. You should also pay more attention to the following action
7761 rules:

7762 1. If "<\$success_detection\$" means the overall task was successful
7763 or equal to "True", then output action MUST be empty like ''. Be
7764 careful to check the task was really successful.

7765 2. You should output actions in Python code format and specify any
7766 necessary parameters to execute that action. Only use function names
7767 and argument names exactly as shown in the valid actions et. If a
7768 function has parameters, you should also include their names and
7769 decide their values, like "press_shift(duration=1)". If it does not
7770 have a parameter, just output the action, like "release_mouse_buttons
7771 ()".

7772 3. Before typing text, ensure that the last executed action involved
7773 clicking on the relevant input box. If the last action was not a
7774 click on this input box, the required action MUST be to click on the
7775 corresponding input box before proceeding.

7776 4. Given the current situation and task, you should only choose the
7777 most suitable action from the valid action set. If values in the
7778 screen are already correct, no need for a new action.

7779 5. When you decide to perform a mouse action, if there is bounding
7780 box in the current screenshot, you MUST choose skill click_on_label(
7781 label_id, mouse_button).

7782 6. When you perform a mouse action, always select the target UI
7783 element closest to the UI element of the previous action for
7784 operation.

7776 7. When you decide to operate on a file, such as downloading it,
7777 please pay attention to the path and name of the current file.
7778 8. If upon self-reflection you think the target coordinates were an
7779 issue, you MUST pay close attention to choosing new coordinates that
7780 are not the same or too similar to the previous ones.
7781 9. If upon self-reflection you think the last action was unavailable
7782 at the current state, you SHOULD try to take another action to try to
7783 enable the desired action.
7784 10. If you leave the application incorrectly, you can go back to it
7785 directly using `go_back_to_target_application()`. No need to use the
7786 mouse.

7787 You should only respond in the format described below. In your reasoning
7788 for the chosen actions, also describe which item you decided to
7789 interact with and why. DO NOT change the title of each item. You
7790 should not output other comments or information besides the format
7791 below:

7792 Decision_Making_Reasoning:
7793 1. ...
7794 2. ...
7795 3. ...

7796 Actions:
7797 ```python
7798 action(args1=x,args2=y)
7799 ```

7800 Key_reason_of_last_action:
7801 ...
7802
7803
7804
7805
7806
7807
7808
7809
7810
7811
7812
7813
7814
7815
7816
7817
7818
7819
7820
7821
7822
7823
7824
7825
7826
7827
7828
7829