

000 001 002 003 004 005 MORTAR: EVOLVING MECHANICS FOR 006 AUTOMATIC GAME DESIGN 007 008 009

010 **Anonymous authors**
011 Paper under double-blind review
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026

ABSTRACT

011 We present MORTAR, a system for autonomously evolving game mechanics for
012 automatic game design. Game mechanics define the rules and interactions that
013 govern gameplay, and designing them manually is a time-consuming and expert-
014 driven process. MORTAR combines a quality-diversity algorithm with a large
015 language model to explore a diverse set of mechanics, which are evaluated by
016 synthesising complete games that incorporate both evolved mechanics and those
017 drawn from an archive. The mechanics are evaluated by composing complete
018 games through a tree search procedure, where the resulting games are evaluated
019 by their ability to preserve a skill-based ordering over players—that is, whether
020 stronger players consistently outperform weaker ones. We assess the mechanics
021 based on their contribution towards the skill-based ordering score in the game.
022 We demonstrate that MORTAR produces games that appear diverse and playable,
023 and mechanics that contribute more towards the skill-based ordering score in the
024 game. We perform ablation studies to assess the role of each system component
025 and a user study to evaluate the games based on human feedback.
026

027 1 INTRODUCTION 028

029 Procedural content generation (PCG) is a well-studied approach in game design, concerned with the
030 automatic creation of game content such as levels, maps, items and narratives (Shaker et al., 2016;
031 Liu et al., 2021). PCG serves multiple purposes: enabling runtime content generation in games
032 such as roguelikes, providing ideation tools for designers, automating the production of repetitive
033 content, and facilitating research into creativity and design processes. Traditionally, PCG research
034 has focused on structural aspects of games—particularly level or layout generation (Risi & Togelius,
035 2020)—where the goal is to produce environments that are coherent, solvable, and varied.

036 By contrast, comparatively little attention has been paid to the procedural generation of *game mechanics*—
037 the underlying rules for interactions that govern gameplay. Yet mechanics play a central
038 role in shaping the player experience, determining not just how players act, but what kinds of strate-
039 gies and emergent behaviours are possible. Designing mechanics is inherently challenging: unlike
040 levels, which can be evaluated by solvability or novelty, the utility of a mechanic depends on the
041 dynamics it induces within the context of a game. This makes both generation and evaluation sig-
042 nificantly harder.

043 A central premise of this work is that evaluating game mechanics is fundamentally more difficult
044 than evaluating assets or level layouts. Unlike these forms of content, a mechanic *cannot be judged*
045 *in isolation*—it only gains meaning through the gameplay it enables. A mechanic that appears novel
046 or complex may still be uninteresting if it does not support skill-based interaction. This insight
047 motivates our approach: effective automation of mechanic design requires not only a generative
048 model, but also a principled way to assess a mechanic’s utility in the context of play.

049 We address this challenge by introducing a mechanic-centric framework for automatic game design.
050 The central idea is to evolve mechanics not in isolation, but through their contribution to the quality
051 of full games. Specifically, we evaluate mechanics by constructing complete games around them,
052 and measuring whether the resulting games induce a skill-based ordering over players of different
053 capabilities. This allows us to define a concrete notion of usefulness for a mechanic: its contribution
to the overall expressivity and skill gradient of the games in which it appears.

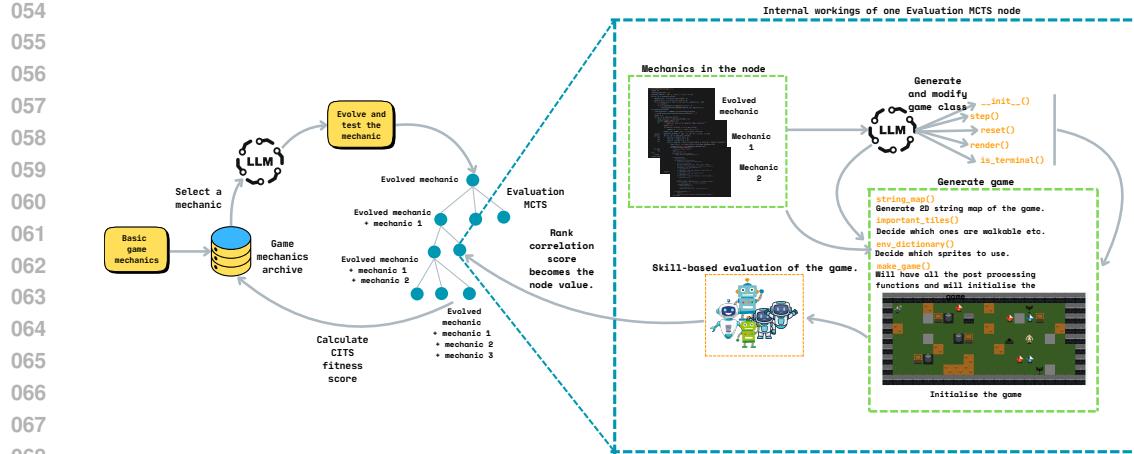


Figure 1: A flow diagram of MORTAR

We introduce MORTAR, a system that evolves game mechanics using a quality-diversity algorithm guided by a large language model (LLM). MORTAR maintains a diverse archive of mechanics, represented as code snippets, which are mutated and recombined through LLM-driven evolutionary operators. Each evolved mechanic is evaluated by embedding it into full games constructed via Monte Carlo Tree Search, which incrementally builds games by composing mechanics from the archive. These games are evaluated based on their ability to induce a consistent skill-based ranking over a fixed set of agents. We define a novel fitness measure, which quantifies the contribution of a mechanic to the final game’s skill-based ordering, inspired by Shapley values (Shapley, 2016).

We demonstrate that MORTAR can evolve a diverse set of game mechanics that contribute to the quality and playability of the generated games.¹ The resulting games exhibit coherent structure, varied interaction patterns, and meaningful skill gradients. Through ablations, we show that both the tree-search-based composition and the LLM-driven mutations are critical for generating high-quality mechanics. Our results highlight the potential for using LLMs not only as generators, but as evaluators and collaborators in the game design loop.

The system described here is a research prototype for the purposes of understanding how to best generate complementary game mechanics. However, it could also serve as an ideation tool for game designers, suggesting new mechanics and mechanic combinations, perhaps in response to designer input. It is not meant to generate complete games, and aims to empower rather than replace game designers.

2 METHOD

MORTAR is an evolutionary algorithm for generating game mechanics, where a large language model (LLM) is used to implement code-level variation operators. A core principle of the method is that a mechanic’s value lies in the gameplay it enables; mechanics are evaluated not in isolation, but by the contribution they make to full games. We formalise this through a notion of *importance*, which guides the search process.

2.1 EVOLUTION SETUP

MORTAR employs a Quality-Diversity (QD) algorithm, using a fixed 2D archive (as in MAP-Elites (Mouret & Clune, 2015)) to store and explore diverse game mechanics. We refer to this structure as the *Mechanics Archive*. Each mechanic is represented as a Python function belonging to a game class, and placed in the archive based on two behavioural descriptors:

¹Play the generated games at: <https://mortar-x3p7.onrender.com/>

108 1. **Mechanic Type:** A categorical descriptor indicating the gameplay behaviour the mechanic
 109 enables. We define 8 mechanic types (detailed in Section 3), each associated with 10 de-
 110 scriptive category words. To classify an evolved mechanic, we compute similarity scores
 111 between the mechanic’s name and all category words, creating a normalised similarity vec-
 112 tor. The mechanic type is determined by identifying the highest similarity score’s index
 113 and multiplying this index by the score to produce a positional similarity value that serves
 114 as the behavioural descriptor.

115 2. **Code Complexity:** Computed using weighted Abstract Syntax Tree (AST) analysis. We
 116 parse the mechanic’s code into an AST representation and calculate complexity as a
 117 weighted sum of function calls, assignments, and return statements. Function calls receive
 118 the highest weight, as mechanisms requiring more function calls exhibit greater complexity.
 119 Assignments are weighted to reflect that additional variables may enable more interesting
 120 behaviours. Return statements contribute to complexity scoring because multiple exit paths
 121 can produce diverse behavioural outcomes.

122 Mechanics are selected from the archive and modified using several LLM-implemented evolutionary
 123 operators: *Mutation* adds new functionality to a single mechanic; *diversity mutation* samples three
 124 mechanics and prompts the LLM to generate a behaviorally distinct variant; *crossover* merges two
 125 mechanics (selected based on AST similarity) into a functional combination that integrates elements
 126 from both; and *compatibility mutation* generates mechanics that complement existing ones in a
 127 game, primarily used during game evaluation (see subsequent sections).

128 2.2 EVALUATING GAME MECHANICS

131 Each evolved mechanic is represented as a function within a Python class. To prepare it for evalua-
 132 tion, we prompt the LLM to construct the rest of the class around it in a step-by-step fashion, starting
 133 with the `__init__()` method to define any required variables and scaffolding, `step` method to add
 134 actions, `reset` and `render` method as needed.

135 The mechanic is then tested for syntax and runtime errors. If no errors occur, we simulate it in a
 136 static test environment with simple objects and characters for the mechanic to interact with them,
 137 if necessary. A Monte Carlo Tree Search (MCTS) agent is used to interact with the environment,
 138 verifying that the mechanic is functional and non-trivial. Only mechanics that pass both tests proceed
 139 to the *usefulness* evaluation stage. Failed mechanics are discarded to reduce unnecessary LLM calls.

140 2.3 AUTOMATED GAME CONSTRUCTION

142 To evaluate a mechanic’s usefulness, we embed it within a full game. Games are constructed through
 143 MCTS, where the root node is the evolved mechanic, and each expansion adds a new mechanic that
 144 is either sampled from the archive or generated via compatibility mutation. Each path through the
 145 tree represents a particular combination of mechanics; that is, a complete game.

146 Games are also implemented as Python classes, following a common template with core methods,
 147 such as `step`, `reset`, `render`, move mechanics and preset variables. The LLM is prompted to
 148 modify or add functionality to these methods as needed, in an iterative manner. It is also asked to
 149 define any helper methods or variables required by the mechanics. At the end of this process, the
 150 LLM is prompted to define a win condition and generates a corresponding termination function. It
 151 also selects appropriate tiles from a predefined set, maps them to characters, and generates a 2D
 152 string-based level layout using these mappings. A final function defines which tiles are walkable,
 153 interactive, or character-specific.

154 The complete game script includes the game class, the tile and map generation functions, and a
 155 preset function that instantiates the full game. Simple postprocessing ensures the map is rectangular,
 156 contains exactly one player, and is free of formatting issues (e.g. whitespace padding).

158 2.4 EVALUATION OF THE GAME

160 A central idea in our work is that a game’s quality is revealed through the emergence of a consistent
 161 skill gradient or game depth: a well-designed game should allow players of differing abilities to be
 meaningfully distinguished. We implement this by evaluating how well the game ranks a fixed set

162 of players by skill. This approach allows us to evaluate not just whether a game is playable, but
 163 whether it rewards skill—a more robust signal of design quality.

164 To assess whether a game rewards skill, we fix a pool of five agents with varying ability levels,
 165 inspired by Nielsen et al. (2015). These include three MCTS agents with increasing numbers of
 166 rollouts, a random agent, and an agent that takes no actions.² This defines a clear expected skill
 167 ordering: the strongest agent should be the MCTS variant with the most rollouts, followed by the
 168 medium and low rollout agents, then the random agent, and finally the no-op agent. The outcome
 169 rank is induced by playing the game and recording empirical win rates. To quantify alignment
 170 between the expected and outcome rankings, we compute Kendall’s Tau (τ), a standard measure of
 171 rank correlation: $\tau = \frac{C-D}{p(p-1)}$. Here, C and D are the number of concordant and discordant pairs,
 172 respectively, and p is the number of players (five in this case). Concordance occurs when the relative
 173 ranking between two players agrees between the expected and observed orders; discordance occurs
 174 when they disagree. A value of 1 indicates perfect alignment with the expected ranking, 0 indicates
 175 no correlation, and -1 reflects a completely reversed order. We consider a game unplayable if
 176 $\tau = -1$.

177 While τ provides a global measure of game quality, it reveals nothing about the source of that
 178 quality. To address this, we introduce *Constrained Importance Through Search* (CITS), a scoring
 179 method to measure each mechanic’s marginal contribution to the emergence of a skill gradient.
 180 Inspired by Shapley values Shapley (2016), CITS estimates how much each mechanic contributed
 181 to the final game’s τ score. However, computing full Shapley values would require evaluating every
 182 subset of mechanics—exponential in the number of mechanics. Instead, CITS is defined over the
 183 exploration tree constructed during generation, making it computationally tractable and grounded in
 184 actual gameplay evaluations. Formally, the CITS score for mechanic i is:

$$186 \quad \text{CITS}_i = \frac{1}{|N_i|} \sum_{n \in N_i} \phi_i^{(n)},$$

189 where $N_i = \{n \in T : i \in M_n, n \neq n_{\text{root}}\}$ is the set of non-root nodes in the tree T that contain
 190 mechanic i , and M_n is the mechanic set at node n . The contribution $\phi_i^{(n)}$ is computed using the
 191 standard Shapley formula over the restricted set of explored subsets:

$$194 \quad \phi_i^{(n)} = \sum_{S \subseteq M_n \setminus \{i\}} \frac{|S|! \cdot (|M_n| - |S| - 1)!}{|M_n|!} \cdot \Delta_i^{(n)}(S),$$

197 where the marginal value term is defined as the difference in value when adding mechanic i to the
 198 subset S :

$$201 \quad \Delta_i^{(n)}(S) = v_T(S \cup \{i\}) - v_T(S).$$

203 Finally, the value function $v_T(S)$ returns the τ_m score for the node m with exactly mechanics S , if
 204 such a node exists in the tree; otherwise, it is defined to be 0:

$$207 \quad v_T(S) = \begin{cases} \tau_m & \text{if } \exists m \in T \text{ s.t. } M_m = S \\ 0 & \text{otherwise} \end{cases}$$

210 This search-constrained Shapley approach allows us to assess a mechanic’s value in context, measur-
 211 ing its contribution within actual, discovered game designs rather than hypothetical combinations.
 212 As such, the CITS score provides a principled, interpretable, and efficient mechanism for attributing
 213 gameplay quality to individual mechanics.

214
 215 ²Any agents with a clear capability ordering would suffice, such as heuristic agents with different search
 depths, or reinforcement learning agents with varying training budgets.

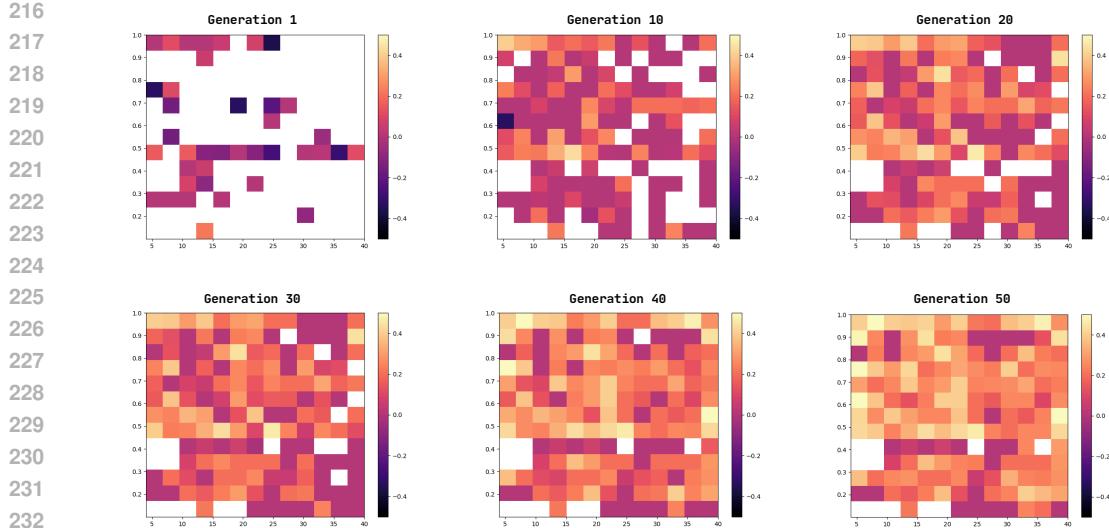


Figure 2: Coverage of game mechanic archive over a run.

3 EXPERIMENT SETUP

MORTAR employs a 2D Quality-Diversity (QD) archive with dimensions for mechanic type (0–1.0) and code complexity (4–40), forming a 13×13 grid. The first dimension categorises mechanics into nine types: *movement, interaction, combat, progression, environment, puzzle, resource management, exploration, time manipulation*. To categorise a mechanic, we use DistilBERT (Sanh et al., 2019) embeddings to compute the similarity between mechanic function names and associated category words (detailed in the Appendix C). The complexity dimension and archive ranges were determined through experimentation to maximise archive coverage.

The system operates with a batch size of 10, selecting individuals from the archive and applying evolutionary operators in parallel. Operator selection probabilities are 50% for diversity mutation, 30% for mutation, and 20% for crossover. Diversity mutation samples three mechanics, while crossover selects pairs based on AST similarity. The static environment used to evaluate the evolved mechanic in isolation can be found in the Appendix A.

For game construction, we use MCTS with 20 iterations, where each expansion adds one mechanic per node (maximum 3 children per node). Unlike traditional MCTS, we do not simulate; instead, we evaluate the complete game formed by all mechanics on the path from root to the newly expanded node, then backpropagate before proceeding to the next expansion. Compatibility mutation generates new mechanics within nodes, with a 50% probability of creating novel mechanics versus selecting from the existing archive. All LLM operations use GPT-4o-mini for both evolution and game creation. Skill assessment employs five agents with a clear capability ordering: MCTS variants with 100,000, 10,000, and 1,000 iterations, plus random and no-action agents for Kendall’s Tau rank correlation computation.

We conduct extensive ablation studies, replacing the MCTS procedure with three alternatives: random mechanic selection, LLM-prompted selection, and greedy fitness-based selection. Each method generates games with 1–4 mechanics to compute CITS scores. We then conduct another ablation with a *Sokoban* (Murase et al., 1996) level as the initial game. All experiments are averaged over five runs due to computational constraints (approximately \$30–50 per run with GPT-4o-mini).

Our evaluation metrics assess MORTAR’s progression through multiple measures. The Quality-Diversity (QD) Score sums all fitness values to indicate improving mechanic quality. Accumulated Rank Correlation totals Kendall’s τ scores across all tree nodes. We track both maximum and mean fitness scores via CITS evaluation, monitor the number of elites filling the archive, and calculate game creation success rate as the proportion of functional games among all generated games.

Furthermore, a user study is conducted to get feedback to known if the games are actually interesting or not. We provide 6 generated games, and pair them together according to their distribution. We then ask the user to play the games and mark which one of the two is more *interesting, novel, fun to play, and easy to understand*. We also give them an option of *Neither*, which is very important to us as it will let us know if the games are actually meaningful.

4 RESULTS

In this section, we analyse results from the complete MORTAR pipeline, ablation and user studies. The QD score, which sums fitnesses of all archive individuals, demonstrates MORTAR’s ability to evolve increasingly better mechanics over time (Figure 3a). Figure 3b reveals complementary patterns: mean fitness (CITS score) increases gradually across generations while maximum fitness shows stepwise improvements, indicating MORTAR’s capacity for continued mechanic discovery. Figure 3c provides additional evidence of progression through the accumulative Kendall’s τ rank correlation score per Evaluation MCTS tree, showing that MORTAR increasingly identifies engaging games that exhibit meaningful skill-based player rankings across generations.

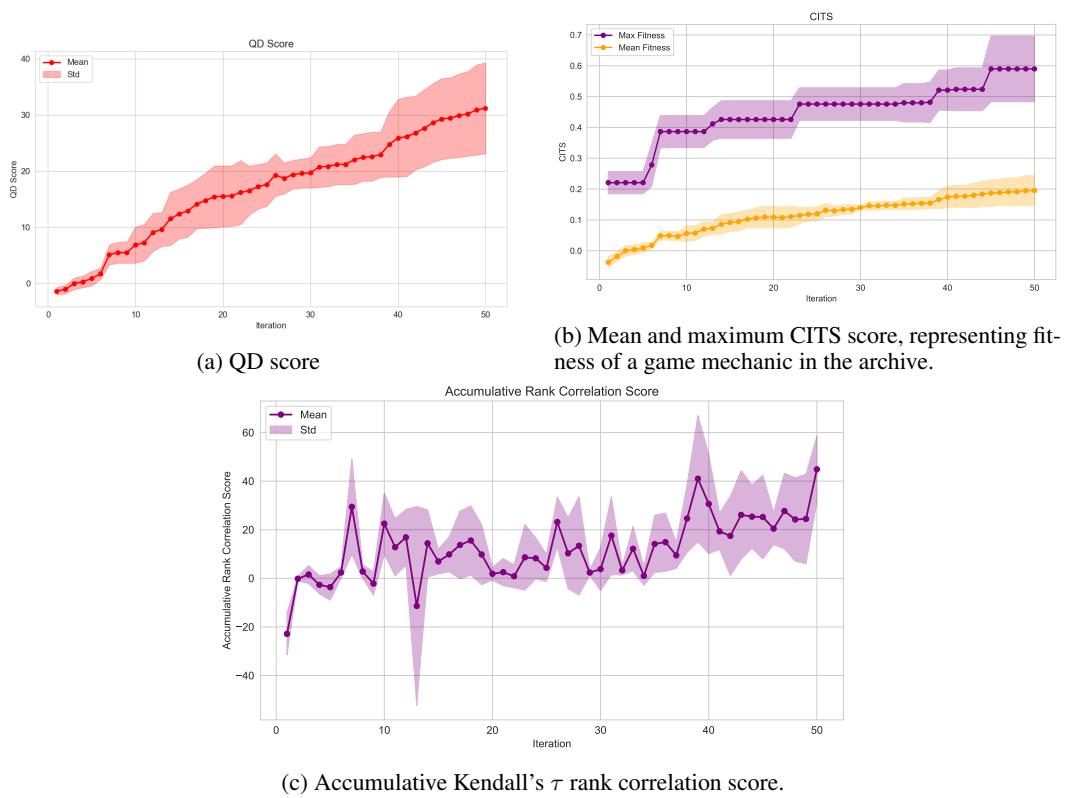


Figure 3: Performance metrics over evolutionary generations.

Table 1 compares MORTAR with alternative approaches to the MCTS evaluation component, our core methodological contribution. MORTAR demonstrates superior evolvability through the highest archive coverage and consistently achieves the best QD score, maximum CITS score, and mean CITS score, indicating its ability to discover higher-quality mechanics. While Greedy Search achieves a marginally better game creation success rate—likely because it always selects the most fit mechanics—this suggests that highly fit mechanics have greater potential for generating playable games. However, MORTAR’s comprehensive performance across multiple metrics demonstrates the effectiveness of its tree search-based composition approach for mechanic evolution. Furthermore, *Sokoban Initialisation* suggests that the MORTAR is sensitive to the initial mechanics and game layout, which impacts evolvability, as evidenced by the very low number of elites in this case.

324 Table 1: Comparison of MORTAR with alternative mechanic selection strategies: LLM-based selec-
 325 tion, random selection, and greedy fitness-based selection across quality-diversity metrics.
 326

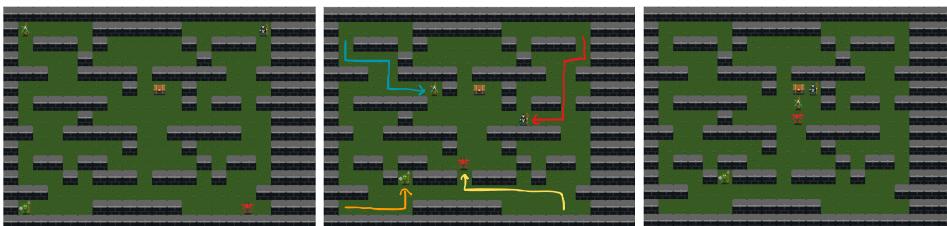
327 Method	No. of elites \uparrow	QD score \uparrow	Max CITS \uparrow	Mean CITS \uparrow	Games success rate \uparrow
328 Evaluation MCTS (ours)	155 \pm 4.51	31.18 \pm 8.10	0.59 \pm 0.11	0.20 \pm 0.05	16.97 \pm 4.64
329 LLM Selection	141 \pm 5.83	17.64 \pm 4.91	0.27 \pm 0.09	0.13 \pm 0.06	11.69 \pm 5.14
330 Random Selection	144 \pm 9.10	9.86 \pm 6.71	0.14 \pm 0.08	0.06 \pm 0.04	11.77 \pm 4.19
331 Greedy Selection	139 \pm 5.15	25.37 \pm 2.81	0.51 \pm 0.06	0.18 \pm 0.13	18.24 \pm 1.19
332 Sokoban Initialisation	110 \pm 10.52	15.19 \pm 3.12	0.45 \pm 0.12	0.19 \pm 0.07	15.11 \pm 2.83

333
 334 Figures 4 and 5 showcase two games generated by MORTAR, demonstrating diversity in level lay-
 335 outs, win conditions, and mechanics. *AllyCraft* (Figure 4) presents a challenging strategic experience
 336 where players control both their character and summoned allies, with escalating difficulty requiring
 337 versatile tactics. Effective strategies involve summoning allies strategically and eliminating enemies
 338 in optimal sequences. This game achieves a Kendall’s τ of 0.8, maintaining clear agent rankings
 339 despite low overall rewards, with only minor rank switching between the do-nothing and random
 340 agents due to negative scoring.

341 By contrast, *TreasureHunt* (Figure 5) exhibits a Kendall’s τ of 0.4, showing significant rank dis-
 342 tortion except for the top-performing agent. This lower correlation suggests reduced strategic
 343 depth—once players discover the optimal path, the game loses replay value. *AllyCraft*’s higher
 344 τ score correlates with sustained engagement through multiple viable strategies, while *Treasure-
 345 Hunt*’s deterministic solution path limits long-term interest. Both games incorporate sophis-
 346 ticated mechanics, including ally summoning, multi-unit control, and pathfinding algorithms. The complete
 347 evolved code for these mechanics is provided in Appendix B.



348
 349 Figure 4: *AllyCraft* gameplay sequence: (Top left) Initial state showing black-marked enemies to
 350 defeat and items to collect for rewards. (Top centre) Player spawns and controls allies as additional
 351 units. (Top right) Allies collect items while enemies advance each turn. (Bottom left) One ally is
 352 defeated by an enemy while simultaneously eliminating an opposing unit. (Bottom centre) Player
 353 and remaining ally attempt coordinated attack but are overwhelmed by enemies, resulting in a loss.
 354
 355
 356
 357



358
 359 Figure 5: *TreasureHunt* gameplay sequence: (Left) Initial game state showing treasure objective in
 360 a capture-the-flag style layout. (Centre) Player spawns at the top-left corner (blue marker) while
 361 enemies begins pursuit. (Right) Final state showing close competition between player and red-
 362 marked enemy, with victory determined by action processing order. The game features an evolved
 363 A* pathfinding algorithm for enemy movement (code in Appendix B).
 364
 365
 366
 367
 368
 369
 370
 371
 372

378
379

4.1 USER STUDY

380
381
382
383
384

To determine whether the quantitative metrics correlate with human preferences, we conducted a small comparative user study with 10 participants who evaluated 6 games generated by MORTAR across five dimensions: interestingness, novelty, frustration level, fun factor, and ease of understanding. Table 2 presents the results alongside each game’s Kendall’s τ score for comparison with MORTAR’s automated evaluation.

385
386
387
388
389

The study compared three pairs of games: *TreasureHunt* versus *HuntBreakout* (capture-the-flag variants where *HuntBreakout* adds wall-breaking mechanics), *AllyCraft* versus *CrystalCavernsCommander* (RPG-style games differing in ally control mechanisms), and *MagneticProwess* versus *HeroHunt* (Sokoban-based games with magnetic pulling and enemy combat mechanics, respectively). See Appendix D for games in the user study.

390
391
392
393
394
395
396

We observe a general correlation between the total human preference score and MORTAR’s calculated Kendall’s τ values. In the first comparison, the τ difference has a smaller magnitude than the total score difference, yet both favour the same game. The second comparison shows alignment in both magnitude and direction between total score and τ . However, the third comparison reveals opposing trends where the total score contradicts τ , though this discrepancy may reflect the inherent difficulty of aligning automated skill-based metrics with subjective human preferences across diverse game genres.

397
398
399
400
401
402
403
404
405
406

Treating the total score as a meaningfulness metric—comprising interestingness, novelty, fun factor, ease of understanding, minus frustration—the “Neither” votes provide additional insight. These scores (1, 7, and 2 across the three comparisons, respectively) indicate that games in the second comparison are perceived as less meaningful, likely due to excessive complexity. This finding aligns with intuitive game design principles: mini-games benefit from appropriate rather than maximal complexity. While complexity can enhance engagement in full games through progressive difficulty scaling, these mini-game experiences demonstrate reduced effectiveness when sophisticated mechanics overwhelm fundamental gameplay elements. Finally, qualitative participant feedback consistently highlighted visual limitations, particularly the absence of animations and restricted sprite sets—a known limitation of MORTAR’s current implementation.

407
408
409

Table 2: User study results comparing games. Values indicate the number of participants (out of 10) selecting each option. Total score represents the sum of positive metrics minus “Frustrating”.

Games	Interesting ↑	Novel ↑	Frustrating ↓	Fun to play ↑	Easy to understand ↑	Total ↑	τ ↑
<i>TreasureHunt</i>	0	1	3	1	4	3	0.4
<i>HuntBreakout</i>	8	8	5	7	4	22	0.5
Both	1	0	0	1	2	2	—
Neither	1	1	2	1	0	1	—
<i>AllyCraft</i>	7	6	5	6	3	17	0.8
<i>CrystalCavernsCommander</i>	2	2	3	3	2	6	0.3
Both	0	1	2	0	1	0	—
Neither	1	1	0	1	4	7	—
<i>MagneticProwess</i>	4	4	5	4	3	10	0.6
<i>HeroHunt</i>	5	5	2	5	3	16	0.3
Both	0	0	1	0	3	2	—
Neither	1	1	2	1	1	2	—

421

422
423

5 RELATED WORK

424
425
426
427
428
429
430
431

The core focus of MORTAR is evolving game mechanics to serve as an ideation and prototyping tool for game designers and generate novel games for testing learning algorithms. This research falls under Automatic Game Design (AGD), pioneered by (Nelson & Mateas, 2007), who formalized game mechanics through WordNet to generate micro games. Browne (2008) and Togelius & Schmidhuber (2008) independently proposed evolutionary approaches to AGD across different domains. The latter introduced learnability as a quality criterion, inspiring various approximations of skill differentiation over the years (Nielsen et al., 2015; Khalifa et al., 2017) that influence our current approach. Related concepts include game depth (Lantz et al., 2017) and formalisms for measuring a game’s ability to distinguish among agents (Stephenson et al., 2020).

432 Non-evolutionary AGD approaches include constraint solvers for mechanics generation (Zook &
 433 Riedl, 2014) and autoencoders for learning and generating mechanics (Rieder, 2018). Recent work
 434 incorporates LLMs into AGD pipelines: ScriptDoctor generates PuzzleScript games (Earle et al.,
 435 2025), while Gavel evolves Ludii games using LLMs and Quality-Diversity algorithms (Todd et al.,
 436 2024). Similar approaches have generated 2-player games using XML-based languages (Jorge &
 437 Antonio J, 2023). MORTAR distinguishes itself by leveraging the full expressiveness of Python code
 438 generation, creating a search space that scales with advancing LLM capabilities.

439 MORTAR also relates to LLM-driven Procedural Content Generation (Togelius et al., 2011; Shaker
 440 et al., 2016; Liu et al., 2021). Early work included Sokoban level generation using GPT-2/3 (Todd
 441 et al., 2023), MarioGPT for Super Mario Bros levels with Novelty Search (Sudhakaran et al.,
 442 2023), and human-in-the-loop GPT-3 fine-tuning (Nasir & Togelius, 2023). Word2World and
 443 Word2Minecraft generate 2D and 3D games with fixed mechanics (Nasir et al., 2024a; Huang,
 444 2025). MORTAR extends this paradigm by generating multiple game aspects, including mechan-
 445 ics and levels.

446 Finally, MORTAR contributes to research on LLMs as evolutionary operators in Quality-Diversity
 447 algorithms like MAP-Elites (Mouret & Clune, 2015). This approach has been applied to robot mor-
 448 phology evolution (Lehman et al., 2023), neural architecture search using CVT-MAP-Elites (Nasir
 449 et al., 2024b), and Ludii game generation (Todd et al., 2024).

451 6 LIMITATIONS

452 While MORTAR successfully generates novel game mechanics and coherent games with semanti-
 453 cally meaningful CITS scores, several limitations warrant future investigation. The system currently
 454 modifies game rendering functions without incorporating animations, limiting visual richness. Our
 455 experiments used a relatively modest LLM (GPT-4o-mini); stronger models could potentially yield
 456 more sophisticated mechanics and improved code quality. The current 2D top-down perspective
 457 constrains the search space—extending to 3D environments would significantly expand creative
 458 possibilities.

459 Archive initialisation presents another challenge, as improved seeding strategies could enhance con-
 460 vergence and final quality. Similarly, increasing MCTS iterations during evaluation might produce
 461 higher-quality games at the cost of computational resources. Perhaps most significantly, MORTAR’s
 462 autonomous evolution lacks designer control mechanisms. A controllable variant that accepts de-
 463 sign constraints or preferences could better serve as an ideation tool, allowing game developers to
 464 guide the search toward specific gameplay goals while maintaining the system’s creative discovery
 465 capabilities.

466 7 CONCLUSION AND FUTURE DIRECTIONS

467 We present MORTAR, a novel system for generating games through mechanic evolution. MORTAR
 468 combines MAP-Elites, a Quality-Diversity algorithm, with LLM-driven code-level mechanic
 469 evolution. The system evaluates mechanics through MCTS, which constructs complete games in
 470 each tree node and assesses them using skill-based ranking. We introduce the Constrained Im-
 471 portance Through Search (CITS) score, derived from Shapley values, which quantifies a mechanic’s
 472 contribution within the actually searched combination space rather than hypothetical alternatives.

473 Our quantitative results demonstrate MORTAR’s high evolvability and progressive improvement
 474 across generations through comprehensive ablation studies. Qualitative analysis reveals that games
 475 with higher scores exhibit greater strategic depth and complexity, while MORTAR consistently pro-
 476 duces diverse gaming experiences with sophisticated mechanic interactions.

477 MORTAR offers several promising research directions. As an ideation tool, it could support game de-
 478 signers by suggesting novel mechanic combinations responsive to design constraints. The system’s
 479 scalability suggests that initialisation with extensive mechanic libraries and extended evolution pe-
 480 riods could explore previously undiscovered regions of game design space. The generated games
 481 provide rich environments for testing generalisation in reinforcement learning agents (Sutton et al.,
 482 1999), offering diverse challenges with measurable skill gradients.

486 REFERENCES
487

488 Cameron Bolitho Browne. *Automatic generation and evaluation of recombination games*. PhD
489 thesis, Queensland University of Technology, 2008.

490 Sam Earle, Ahmed Khalifa, Muhammad Umair Nasir, Zehua Jiang, Graham Todd, Andrzej
491 Banburski-Fahey, and Julian Togelius. Scriptdoctor: Automatic generation of puzzlescript games
492 via large language models and tree search. *arXiv preprint arXiv:2506.06524*, 2025.

493 Shuo Huang. Word2minecraft: Generating 3d game levels through large language models. Master’s
494 thesis, New York University Tandon School of Engineering, 2025.

495 Ruiz-Quiñones Jorge and Fernández-Leiva Antonio J. Automated videogame mechanics generation
496 with xvgdl. *ICGA Journal*, 44(4):124–152, 2023.

497 Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. General video
498 game rule generation. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*,
499 pp. 170–177. IEEE, 2017.

500 Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. Depth in strategic
501 games. In *AAAI Workshops*, 2017.

502 Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley.
503 Evolution through large models. In *Handbook of evolutionary machine learning*, pp. 331–366.
504 Springer, 2023.

505 Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian
506 Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*,
507 33(1):19–37, 2021.

508 Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint*
509 *arXiv:1504.04909*, 2015.

510 Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. Automatic making of sokoban problems.
511 In *Pacific Rim International Conference on Artificial Intelligence*, pp. 592–600. Springer, 1996.

512 Muhammad U Nasir and Julian Togelius. Practical pcg through large language models. In *2023*
513 *IEEE Conference on Games (CoG)*, pp. 1–4. IEEE, 2023.

514 Muhammad U Nasir, Steven James, and Julian Togelius. Word2world: Generating stories and
515 worlds through large language models. *arXiv preprint arXiv:2405.06686*, 2024a.

516 Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn.
517 Llmatic: neural architecture search via large language models and quality diversity optimization.
518 In *proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1110–1118, 2024b.

519 Mark J Nelson and Michael Mateas. Towards automated game design. In *Congress of the Italian*
520 *Association for Artificial Intelligence*, pp. 626–637. Springer, 2007.

521 Thorbjørn S Nielsen, Gabriella AB Barros, Julian Togelius, and Mark J Nelson. General video
522 game evaluation using relative algorithm performance profiles. In *European Conference on the*
523 *Applications of Evolutionary Computation*, pp. 369–380. Springer, 2015.

524 Bernhard Rieder. Using procedural content generation via machine learning as a game mechanic.
525 *Austrian Marshall Plan Foundation*, 2018.

526 Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural
527 content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.

528 Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of
529 bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

530 Noor Shaker, Julian Togelius, and Mark Nelson. *Procedural Content Generation in Games*.
531 Springer, 2016.

540 Lloyd Shapley. 17. a value for n-person games. In *Contributions to the Theory of Games, Volume*
541 *II*, pp. 307–318. Princeton University Press, 2016.

542

543 Matthew Stephenson, Damien Anderson, Ahmed Khalifa, John Levine, Jochen Renz, Julian To-
544 gelius, and Christoph Salge. A continuous information gain measure to find the most discrim-
545 inatory problems for ai benchmarking. In *2020 IEEE Congress on Evolutionary Computation*
546 (*CEC*), pp. 1–8. IEEE, 2020.

547

548 Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro,
549 and Sebastian Risi. Mariogpt: Open-ended text2level generation through large language models.
550 *Advances in Neural Information Processing Systems*, 36:54213–54227, 2023.

551

552 Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuro-*
553 *science*, 11(1):126–134, 1999.

554

555 Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius.
556 Level generation through large language models. In *Proceedings of the 18th International Con-*
557 *ference on the Foundations of Digital Games*, pp. 1–8, 2023.

558

559 Graham Todd, Alexander G Padula, Matthew Stephenson, Éric Piette, Dennis J Soemers, and Julian
560 Togelius. Gavel: Generating games via evolution and language models. *Advances in Neural*
561 *Information Processing Systems*, 37:110723–110745, 2024.

562

563 Julian Togelius and Jurgen Schmidhuber. An experiment in automatic game design. In *2008 IEEE*
564 *Symposium On Computational Intelligence and Games*, pp. 111–118. IEEE, 2008.

565

566 Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based
567 procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational*
568 *Intelligence and AI in Games*, 3(3):172–186, 2011.

569

570 Alexander Zook and Mark Riedl. Automatic game design via mechanic generation. In *Proceedings*
571 *of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

A PROMPTS

A.1 GAME MECHANIC GENERATION PROMPTS

Prompts provided to MORTAR are accompanied by predefined Python code, which is then modified as required. The following methods of a class are defined separately so they can be invoked in individual prompts:

```
577 1
578 2 init_method = r"""def __init__(self, walkable_tiles,tiles_without_char,
579 3 tiles, str_map_without_chars, str_map, interactive_object_tiles,
580 4 enemy_tiles, render_mode="human"):
581 5     super(GameMechEnv, self).__init__()
582 6     self.map_str_without_chars = str_map_without_chars.strip().split('\n'
583 7 ')
584 8     self.map_str = str_map.strip().split('\n')
585 9     self.map = [list(row) for row in self.map_str]
586 10    self.map_without_chars = [list(row) for row in self.
587 11    map_str_without_chars]
588 12    self.tiles = tiles
589 13    self.tiles_without_char = tiles_without_char
590 14    self.action_space = spaces.Discrete(self.get_action_space())
591 15    self.char_set = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'O': 4, '@': 5, '#':
592 16    6, '&': 7}
593 17    self.char_to_int = lambda c: self.char_set.get(c, 0)
594 18    self.mechanic_to_action = self.get_mechanics_to_action()
595 19    self.done = False
596 20    self.tile_size = 16
597 21    self.char_tile_size = 16
```

```

594    17     self.frames = []
595    18     max_width = max(len(row) for row in self.map_str)
596    19     self.observation_space = spaces.Box(
597    20         low=0,
598    21         high=1,
599    22         shape=(len(self.char_set), len(self.map_str), max_width), # Use
600    23         len(self.char_set) for channels
601    24         dtype=np.int32
602    25     )
603    26
604    27     self.default_walkable_tile = 'A'
605    28     self.render_mode = "rgb_array"
606    29     self.walkable_tiles = walkable_tiles
607    30     self.interactive_object_tiles = interactive_object_tiles
608    31     self.enemy_tiles = enemy_tiles
609    32     self.picked_objects = []
610    33     self.npc_tiles = ["&"]
611    34     self.enemy_tiles = ["#"]
612    35     self.player_health = 100
613    36     self.enemy_health = 100
614    37     self.current_score = 0
615    38     self.map = [list(row) for row in self.map_str]
616    39     self.grid_width = max(len(row) for row in self.map)
617    40     self.grid_height = len(self.map)
618    41     for i, row in enumerate(self.map):
619    42         for j, tile in enumerate(row):
620    43             if tile == '@':
621    44                 self.player_position = (i, j)
622    45
623    46     self.reset() """
624    47     reset_method = """
625    48         def reset(self, seed=None):
626    49             self.map = [list(row) for row in self.map_str]
627    50             self.map_without_chars = [list(row) for row in self.
628    51                 map_str_without_chars]
629    52             self.grid_width = max(len(row) for row in self.map)
630    53             self.grid_height = len(self.map)
631    54             for i, row in enumerate(self.map):
632    55                 for j, tile in enumerate(row):
633    56                     if tile == '@':
634    57                         self.player_position = (i, j)
635    58                     self.current_tile = self.map_without_chars[self.player_position[0]][
636    59                         self.player_position[1]] # Set current tile to the player's starting
637    60                         position
638    61             return self.get_state()["map"] """
639    62
640    63     render_method = """
641    64         def render(self, mode='human'):
642    65             env_img = Image.new('RGBA', (len(self.map[0]) * self.tile_size, len(
643    66                 self.map) * self.tile_size))
644    67
645    68             # 1st layer: Default walkable tile
646    69             for i in range(len(self.map)):
647    70                 for j in range(len(self.map[0])):
648    71                     tile_img = self.tiles[self.default_walkable_tile].resize((
649    72                         self.tile_size, self.tile_size))
650    73                     env_img.paste(tile_img, (j * self.tile_size, i * self.
651    74                         tile_size), tile_img)
652    75
653    76             # 2nd layer: Map without characters
654    77             for i, row in enumerate(self.map_without_chars):
655    78                 for j, tile in enumerate(row):
656    79                     if tile in self.tiles and tile != self.default_walkable_tile:
657    80                         tile_img = self.tiles[tile].resize((self.tile_size, self.
658    81                             tile_size)))

```

```

648      74         env_img.paste(tile_img, (j * self.tile_size, i * self.
649      75             tile_size), tile_img)
650      75
651      76     # 3rd layer: Characters and objects
652      77     for i, row in enumerate(self.map):
653      78         for j, tile in enumerate(row):
654      79             if tile in self.tiles and tile not in self.walkable_tiles:
655      80                 if tile.isalpha():
656      81                     tile_img = self.tiles[tile].resize((self.tile_size,
657      82             self.tile_size))
658      83             else:
659      84                 tile_img = self.tiles[tile].resize((self.
660      85                     char_tile_size, self.char_tile_size))
661      86                     # Center the character in the tile
662      87                     x_offset = (self.tile_size - self.char_tile_size) //
663      88                         2
664      89                     y_offset = (self.tile_size - self.char_tile_size) //
665      90                         2
666      91                     env_img.paste(tile_img, (j * self.tile_size +
667      92             x_offset, i * self.tile_size + y_offset), tile_img)
668      93
669      93     frame = np.array(env_img.convert('RGB'))
670      94     self.frames.append(frame)
671      95     return frame"""
672      96
673      97     get_action_space_method = """def get_action_space(self):
674      98         return 3"""
675      99
676     100
677     101     get_mechanics_to_action_method = """def get_mechanics_to_action(self):
678     102         :
679     103             return {
680     104                 "move_player": 0, # 0-3 for movement
681     105             }"""
682     106
683     107     move_player = """def move_player(self, action):
684     108         moves = {0: (-1, 0), 1: (1, 0), 2: (0, -1), 3: (0, 1)} # Up, Down,
685     109             Left, Right
686     110         dx, dy = moves[action]
687     111         new_row = self.player_position[0] + dx
688     112         new_col = self.player_position[1] + dy
689     113         reward = 0
690     114         if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map[0]):
691     115             new_tile = self.map[new_row][new_col]
692     116             if new_tile in self.walkable_tiles:
693     117                 self.update_player_position(new_row, new_col, new_tile)
694     118         return reward"""
695     119
696     120     other_methods = r"""def update_player_position(self, new_row, new_col,
697     121             new_tile):
698     122         # Validate both current and new positions are within bounds
699     123         if not (0 <= new_row < self.grid_height and 0 <= new_col < self.
700     124             grid_width):
701     125                 return
702     126
703     127         if not (0 <= self.player_position[0] < self.grid_height and 0 <= self
704             .player_position[1] < self.grid_width):
705     128             # If current position is invalid, just set the new position
706     129             self.player_position = (new_row, new_col)
707     130             self.current_tile = new_tile
708     131             self.map[new_row][new_col] = '@'
709     132             return
710     133
711     134         if new_tile not in self.walkable_tiles:
712     135             return
713     136
714     137

```

```

702 128     # Reset the player's previous position to the original tile
703 129     self.map[self.player_position[0]][self.player_position[1]] = self.
704 130     current_tile
705 130     self.map_without_chars[self.player_position[0]][self.player_position
706 131     [1]] = self.current_tile
707 131
708 132     # Update the player's position
709 133     self.player_position = (new_row, new_col)
710 134     self.current_tile = new_tile
711 135     self.map[new_row][new_col] = '@'
712 136
713 137 def find_player_position(self):
714 138     for i, row in enumerate(self.map):
715 139         for j, tile in enumerate(row):
716 140             if tile == '@':
717 141                 return (i, j)
718 142
719 143     return None
720 144
721 145 def clone(self):
722 146
723 147     new_env = GameMechEnv(
724 148         walkable_tiles=self.walkable_tiles,
725 149         tiles_without_char=self.tiles_without_char,
726 150         tiles=self.tiles,
727 151         str_map_without_chars='\n'.join(self.map_str_without_chars),
728 152         str_map='\n'.join(self.map_str),
729 153         interactive_object_tiles=self.interactive_object_tiles,
730 154         enemy_tiles=self.enemy_tiles
731 155     )
732 156     new_env.map = [row[:] for row in self.map]
733 157     new_env.map_without_chars = [row[:] for row in self.map_without_chars]
734 158
735 159     new_env.player_position = self.player_position
736 160     new_env.current_tile = self.current_tile
737 161     new_env.char_to_int = self.char_to_int
738 162     new_env.char_set = self.char_set
739 163     return new_env
740 164
741 165 def is_terminal(self):
742 166     return self.done"""
743 167
744 168     get_state_method = """def get_state(self):
745 169         return {"map": self.map}"""
746 170
747 171     step_method = """def step(self, action):
748 172         reward = 0
749 173         self.done = False
750 174         if action < 4: # Movement actions
751 175             self.move_player(action)
752 176         self.done = reward > 0
753 177         info = {}
754 178         return self.get_state()["map"], reward, self.done, False, info
755 179 """

```

748 The following are the prompts used to edit the methods:

749

750 1. The state, render, and step method prompts are identical, except that the input function is
751 replaced by the function currently in focus:

752

```

1     "Given the following get_state method of the class:\n" +
2     get_state_method + "\n And the following game mechanic:\n" +
3     mechanics + "\n Edit get_state method, if required, for the
4     given game mechanic to work. Do not repeat the game mechanic as
5     a method. Only output whole of the edited get_state method and

```

756 if not edited, just output 'False'. Do not output anything
 757 else."
 758

759 2. Adding helper methods:

760 1 "Given the following methods: init method of the class:\n" +
 761 init_method + "\n All functions already present in the class:\n"
 762 + other_methods + "\n The following game mechanic you just
 763 created:\n" + mechanics + "\n Add any new helper methods needed
 764 for the " + extract_function_name(mechanics) + " to work, if
 765 required. Do not create or update __init__, reset, step,
 766 get_state, render, get_action_space, or get_mechanics_to_action
 767 methods. Do not repeat the game mechanic as a method or the
 768 present methods. Do not add any new variables. Only output the
 769 additional new method or methods required, and if not added,
 770 just output 'False'. Do not output anything else."
 771

772 3. The prompt to add new variables. game_mech_class one string with the whole initial
 773 class present in it.

774 1 "Given the following class:\n" + game_mech_class + "\n And the
 775 following game mechanic:\n" + mechanics + "\n Add any new
 776 variables required in the GameMechEnv for the given game
 777 mechanic to work. Always add to the existing code. Only output
 778 the new variables if they are added in a Python dictionary
 779 format and if not added, just output 'False'. The Python
 780 dictionary can look like {'var_name_1': init_value, 'var_name_2'
 781 ': init_value}. The keys should be the string of the variable
 782 name and the values should be the initial values. The values
 783 can never be a new argument to the init method. Do not output
 784 anything else."
 785

786 4. To generate the action space

787 1 "Given the following class:\n" + game_mech_class + "\n Also,
 788 the following game mechanic:\n" + mechanics + "\n And the
 789 following get_action_space method:\n" + action_space + "\n .
 790 Edit the get_action_space method to add more actions, if
 791 required, for the given game mechanic to work. Do not repeat
 792 the game mechanic as a method. Only output the edited
 793 get_action_space method and if not edited, just output 'False'.
 794 Do not output anything else."
 795

796 5. To get a mapping of mechanics to actions.

797 1 "Given the following class:\n" + game_mech_class + "\n Also,
 798 the following game mechanic:\n" + mechanics + "\n And the
 799 following get_mechanics_to_action method:\n" + mech_to_action +
 800 "\n . Edit the get_mechanics_to_action method to add more
 801 actions, if required, for the given game mechanic to work. The
 802 edition should be "+ extract_function_name(mechanics) +":
 803 action_number. Do not repeat the game mechanic as a method.
 804 Only output the edited get_mechanics_to_action method and if
 805 not edited, just output 'False'. Do not output anything else."
 806

807 6. Game mechanics are then tested on a static environment:

808 1 str_world = """BBBBBBBBBBBB
 809 2 BAAAAAAAB
 810 3 BAAAOAAAAAB
 811 4 BA#@OOAAAAB
 812 5 BA#AAAAAAAB
 813 6 BBBBBBBBBBBB"""
 814
 815 7
 816 8 str_map_wo_chars = """BBBBBBBBBBBB
 817 9 BAAAAAAAB

```
810     10 BAAOOAAAAAB
811     11 BAAAOAAAAAB
812     12 BAAAAAAAAB
813     13 BBBB BBBB BBBB """"
814
815     14
816     15 walkables = ['A', 'B']
817     16 interactive_object_tiles = ['O']
818     17 enemy_tiles = ["#"]
819     18 npc_tiles = ["&"]
820     19 env_image = dict()
821
822     20
823     21
824     22 env_image["A"] = Image.open(r"world_tileset_data/
825         td_world_floor_grass_c.png").convert("RGBA")
826     23 env_image["B"] = Image.open(r"world_tileset_data/
827         td_world_wall_stone_h_a.png").convert("RGBA")
828     24 env_image["C"] = Image.open(r"world_tileset_data/
829         td_world_floor_grass_c.png").convert("RGBA")
830     25 env_image["O"] = Image.open(r"world_tileset_data/td_world_chest.
831         png").convert("RGBA")
832     26 env_image["@"] = Image.open(r"character_sprite_data/
833         td_monsters_archer_d1.png").convert("RGBA")
834     27 env_image["#"] = Image.open(r"character_sprite_data/
835         td_monsters_witch_d1.png").convert("RGBA")
836     28 env_image["&"] = Image.open(r"character_sprite_data/
837         td_monsters_goblin_captain_d1.png").convert("RGBA")
838
839     29
840     30 env = GameMechEnv(walkable_tiles=walkables,
841             tiles_without_char=str_map_wo_chars,
842             tiles=env_image,
843             str_map_without_chars=str_map_wo_chars,
844             str_map=str_world,
845             interactive_object_tiles=interactive_object_tiles,
846             enemy_tiles=enemy_tiles)
```

A.2 GAME GENERATION PROMPTS

All of the above prompts are also used in the game-generation pipeline. The difference is that we keep track of the previously edited method so that, when the evaluation MCTS tree expands to the next node, the prompt includes methods inherited from earlier nodes. This is necessary because each newly expanded node introduces a new mechanic in addition to all prior mechanics.

After these generations, we generate a `make_game` function. We start with generating a `env_dict_func`, a function that returns a dictionary mapping tiles to their corresponding path functions. We provide the following paths:

```
849
850     1     paths_to_tiles = r'''world_tileset_data/td_items_amulet_gold.png,
851     2             world_tileset_data/td_items_gem_ruby.png,
852     3             world_tileset_data/td_world_crate.png,
853     4             world_tileset_data/tg_world_barrel.png,
854     5             world_tileset_data/tg_world_floor_carpet_d.png,
855     6             world_tileset_data/tg_world_floor_moss_e.png,
856     7             world_tileset_data/tg_world_floor_sand_f.png,
857     8             world_tileset_data/tg_world_floor_panel_steel_c.png,
858     9             character_sprite_data/td_monsters_angel_d2.png,
859    10             character_sprite_data/td_monsters_archer_u2.png,
860    11             character_sprite_data/td_monsters_berserker_d1.png,
861    12             character_sprite_data/td_monsters_demon_l1.png'''
```

And the whole initial env\ dict\ func:

```
862 1     env_dict_func = '''def env_dict():
863 2         env_image = dict()
864 3         image_paths = dict()  # New dictionary to store paths
```

```

864
865     # Define a function to load image and store path
866     def load_image(char, path):
867         env_image[char] = Image.open(path).convert("RGBA")
868         image_paths[char] = path # Store the path
869
870     # Load all images
871     base_path = r"/mnt/lustre/users/mnasir/gmd"
872     load_image("A", f"{base_path}/world_tileset_data/
873     td_world_floor_grass_c.png")
874     load_image("B", f"{base_path}/world_tileset_data/
875     td_world_wall_stone_h_a.png")
876     load_image("X", f"{base_path}/world_tileset_data/
877     td_world_floor_grass_c.png")
878     load_image("O", f"{base_path}/world_tileset_data/td_world_chest.png")
879     load_image("I", f"{base_path}/world_tileset_data/td_world_chest.png")
880     load_image("C", f"{base_path}/world_tileset_data/td_world_chest.png")
881     load_image("@", f"{base_path}/character_sprite_data/
882     td_monsters_archer_d1.png")
883     load_image("#", f"{base_path}/character_sprite_data/
884     td_monsters_witch_d1.png")
885     load_image("&", f"{base_path}/character_sprite_data/
886     td_monsters_goblin_captain_d1.png")
887
888     # Here you add any other tiles that are needed for the game in the
889     same format
890
891     return env_image, image_paths ''

```

889 Therefore, the prompt for the `env\dict\func` generation:

```

890     "Given the game mechanics:\n" + mechanics[0] + "\nChange the
891     following env_dict function to cater for the mechanics, if required
892     .:\n" + env_dict_func + " Use the same paths for the same type of
893     tiles that are being added in env_image, or use the following paths:\n
894     " + paths_to_tiles + "\n You don't have to use the paths provided if
895     not needed. Strictly use the same paths. Do not use any other paths.
896     Only add in env_image if needed. Return the full env_dict function."
897

```

897 Then we create a 2D character map through the prompt:

```

898     "Given the game mechanics:\n" + mechanics[0] + "\n The env_dict
899     function, which has the paths for the tiles:\n" +
900     env_dict_func_changed['choices'][0]['message']['content'] + "\nChange
901     the following str_world in the str_map function to cater for the
902     mechanics, if required.: \n" + str_map_func + "\n 'str_world' is the
903     string that represents the 2D game map. Change it if only needed. It
904     must always have 1 and only 1 '@' character, which represents the
905     player. Return the full str_map function."
906

```

906 where the `str\map\func` is:

```

907     str_map_func = '''def str_map():
908
909         str_world = """AAAAAAAAAAAAAAAAAAAA
910         AAAAAAAA@AAAAAAA
911         AAA@OAXAAAAAAA
912         AAAAAA@CAAAAAAAA
913         AAA#AAAAAAA
914         AAAAAAAA@AAAAA"""
915
916         return str_world'''

```

917 Lastly, in the `make\game` function we generate the `important\tiles\func` through the prompt:

```

918 1 "Given the game mechanics:\n" + mechanics[0] + "\n And the env_dict
919 function, which has the paths for the tiles:\n" +
920 env_dict_func_changed['choices'][0]['message']['content'] + "\n And
921 the str_map function, which has the string representation of the game
922 map:\n" + str_map_func_changed['choices'][0]['message']['content'] +
923 "\n Change the following important_tiles function to cater for the
924 mechanics, if required.:n" + important_tiles_func + " Return the
925 full important_tiles function. Return all the tile types mentioned in
926 the return statement of the function. Return empty list if the tile
927 type is not needed."

```

928 where the initial `important_tiles_func` is:

```

929 1 important_tiles_func = '''def important_tiles():
930 2     walkables = ['A'] # Walkable tiles
931 3     non_walkables = ['B'] # Non-walkable tiles
932 4     interactive_object_tiles = ['O', 'I', 'C'] # Interactive objects
933 (e.g., chests)
934 5     collectible_tiles = [] # Can add collectible tiles if needed
935 6     npc_tiles = [] # Assume there are no NPCs represented in the
936 current setup
937 7     player_tile = ['@'] # Player tile
938 8     enemy_tiles = ['#', '&'] # Enemy tiles
939 9     extra_tiles = [] # any other type of tiles for the game goes here
940 10    return walkables, non_walkables, interactive_object_tiles,
941 collectible_tiles, npc_tiles, player_tile, enemy_tiles,
942 '''

```

942 For game-mechanic generation, the terminal method is fixed, since we test each mechanic in isolation to verify that the MCTS agent can reach the end. In the subsequent game-generation stage, 943 however, the terminal method may vary, as the win condition can change substantially.

```

943 1 "Given the game mechanics:\n" + get_games_scores.latest_methods['
944 mechanic'] + "\n" + mechanics[0] + "\n and the init function:\n" +
945 get_games_scores.latest_methods['init'] + "\n We want to train an
946 agent to play a game that uses these mechanics. The layout of the
947 game is the str_world in the following function:\n" + get_games_scores
948 .latest_methods['str_world'] + "\n and the following function
949 describes what the tiles mean:\n" + get_games_scores.latest_methods['
950 tiles'] + "\n The following line describes the situation of the win
951 condition of the game:\n" + "'"+line_response['choices'][0]['message'
952 ]['content']+"" + "\n Write one method of the class which which wraps
953 win condition in it and tells the agent when the game will end. It
954 must focus on the mechanics in the game provided to you. All the
955 mechanics should be used to fulfill the win condition. The name of
956 the method should be is_terminal. Method should only return one
957 boolean variable. Only return the method and nothing else."

```

958 Here a `line_response` is a win condition generated through the following prompt:

```

959 1 "Given the game mechanics:\n" + get_games_scores.latest_methods['
960 mechanic'] + "\n" + mechanics[0] + "\n write one line that describes
961 the win condition for the game that will use these mechanics. "

```

962 The name of the game is generated through:

```

963 1 "Given the game mechanics:\n" + get_games_scores.latest_methods['
964 mechanic'] + "\n" + mechanics[0] + "\n The win condition for the game
965 in is_terminal method:\n " + is_terminal_response['choices'][0]['
966 message'] ['content'] + "\n And the line that explains the win
967 condition:\n " + line_response['choices'][0]['message'] ['content'] +
968 "\n Give the game a short name that describes the game well. Only
969 strictly output the name and nothing else. Should not have any
970 special characters in the name. Do not highlight the name."

```

972 B GENERATED GAME MECHANICS
973974 B.1 MECHANICS IN FIGURE 6
975976 Following is the mechanic and helper functions for the game in Figure 6:
977

```

978 1 def spawn_unit(self):
979 2     """Spawn a unit at an adjacent empty position"""
980 3     reward = 0
981 4
982 5     if len(self.units) >= self.max_units:
983 6         return 0 # No penalty, just no reward
984 7
985 8     player_row, player_col = self.player_position
986 9     adjacency_offsets = [(0, -1), (0, 1), (-1, 0), (1, 0)] #Left, Right,
987 10    Up, Down
988 11
989 12    # Try to find an empty adjacent position
990 13    for dx, dy in adjacency_offsets:
991 14        new_row = player_row + dx
992 15        new_col = player_col + dy
993 16        if (0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
994 17        [0])):
995 18            # Check the base tile type (without characters)
996 19            base_tile = self.map_without_chars[new_row][new_col]
997 20            current_tile = self.map[new_row][new_col]
998 21
999 22            # Check if position is suitable for unit spawning
1000 23            if (base_tile in self.walkable_tiles and
1001 24                (new_row, new_col) not in self.units and
1002 25                (new_row, new_col) != self.player_position and
1003 26                current_tile not in self.enemy_tiles and
1004 27                current_tile in self.walkable_tiles): # Current tile
1005 28                should also be walkable
1006 29
1007 30                # Spawn unit here
1008 31                unit_pos = (new_row, new_col)
1009 32                self.units.append(unit_pos)
1010 33                self.unit_health[unit_pos] = 100 # Initialize unit
1011 34
1012 35                health
1013 36
1014 37                self.map[new_row][new_col] = self.unit_symbol
1015 38                reward = 1 # Reward for successful spawning
1016 39                break
1017 40
1018 41
1019 42
1020 43
1021 44
1022 45
1023 46
1024 47
1025 48
1026 49
1027 50
1028 51
1029 52
1030 53
1031 54
1032 55
1033 56
1034 57
1035 58
1036 59
1037 60
1038 61
1039 62
1040 63
1041 64
1042 65
1043 66
1044 67
1045 68
1046 69
1047 70
1048 71
1049 72
1050 73
1051 74
1052 75
1053 76
1054 77
1055 78
1056 79
1057 80
1058 81
1059 82
1060 83
1061 84
1062 85
1063 86
1064 87
1065 88
1066 89
1067 90
1068 91
1069 92
1070 93
1071 94
1072 95
1073 96
1074 97
1075 98
1076 99
1077 100
1078 101
1079 102
1080 103
1081 104
1082 105
1083 106
1084 107
1085 108
1086 109
1087 110
1088 111
1089 112
1090 113
1091 114
1092 115
1093 116
1094 117
1095 118
1096 119
1097 120
1098 121
1099 122
1100 123
1101 124
1102 125
1103 126
1104 127
1105 128
1106 129
1107 130
1108 131
1109 132
1110 133
1111 134
1112 135
1113 136
1114 137
1115 138
1116 139
1117 140
1118 141
1119 142
1120 143
1121 144
1122 145
1123 146
1124 147
1125 148
1126 149
1127 150
1128 151
1129 152
1130 153
1131 154
1132 155
1133 156
1134 157
1135 158
1136 159
1137 160
1138 161
1139 162
1140 163
1141 164
1142 165
1143 166
1144 167
1145 168
1146 169
1147 170
1148 171
1149 172
1150 173
1151 174
1152 175
1153 176
1154 177
1155 178
1156 179
1157 180
1158 181
1159 182
1160 183
1161 184
1162 185
1163 186
1164 187
1165 188
1166 189
1167 190
1168 191
1169 192
1170 193
1171 194
1172 195
1173 196
1174 197
1175 198
1176 199
1177 200
1178 201
1179 202
1180 203
1181 204
1182 205
1183 206
1184 207
1185 208
1186 209
1187 210
1188 211
1189 212
1190 213
1191 214
1192 215
1193 216
1194 217
1195 218
1196 219
1197 220
1198 221
1199 222
1200 223
1201 224
1202 225
1203 226
1204 227
1205 228
1206 229
1207 230
1208 231
1209 232
1210 233
1211 234
1212 235
1213 236
1214 237
1215 238
1216 239
1217 240
1218 241
1219 242
1220 243
1221 244
1222 245
1223 246
1224 247
1225 248
1226 249
1227 250
1228 251
1229 252
1230 253
1231 254
1232 255
1233 256
1234 257
1235 258
1236 259
1237 260
1238 261
1239 262
1240 263
1241 264
1242 265
1243 266
1244 267
1245 268
1246 269
1247 270
1248 271
1249 272
1250 273
1251 274
1252 275
1253 276
1254 277
1255 278
1256 279
1257 280
1258 281
1259 282
1260 283
1261 284
1262 285
1263 286
1264 287
1265 288
1266 289
1267 290
1268 291
1269 292
1270 293
1271 294
1272 295
1273 296
1274 297
1275 298
1276 299
1277 300
1278 301
1279 302
1280 303
1281 304
1282 305
1283 306
1284 307
1285 308
1286 309
1287 310
1288 311
1289 312
1290 313
1291 314
1292 315
1293 316
1294 317
1295 318
1296 319
1297 320
1298 321
1299 322
1300 323
1301 324
1302 325
1303 326
1304 327
1305 328
1306 329
1307 330
1308 331
1309 332
1310 333
1311 334
1312 335
1313 336
1314 337
1315 338
1316 339
1317 340
1318 341
1319 342
1320 343
1321 344
1322 345
1323 346
1324 347
1325 348
1326 349
1327 350
1328 351
1329 352
1330 353
1331 354
1332 355
1333 356
1334 357
1335 358
1336 359
1337 360
1338 361
1339 362
1340 363
1341 364
1342 365
1343 366
1344 367
1345 368
1346 369
1347 370
1348 371
1349 372
1350 373
1351 374
1352 375
1353 376
1354 377
1355 378
1356 379
1357 380
1358 381
1359 382
1360 383
1361 384
1362 385
1363 386
1364 387
1365 388
1366 389
1367 390
1368 391
1369 392
1370 393
1371 394
1372 395
1373 396
1374 397
1375 398
1376 399
1377 400
1378 401
1379 402
1380 403
1381 404
1382 405
1383 406
1384 407
1385 408
1386 409
1387 410
1388 411
1389 412
1390 413
1391 414
1392 415
1393 416
1394 417
1395 418
1396 419
1397 420
1398 421
1399 422
1400 423
1401 424
1402 425
1403 426
1404 427
1405 428
1406 429
1407 430
1408 431
1409 432
1410 433
1411 434
1412 435
1413 436
1414 437
1415 438
1416 439
1417 440
1418 441
1419 442
1420 443
1421 444
1422 445
1423 446
1424 447
1425 448
1426 449
1427 450
1428 451
1429 452
1430 453
1431 454
1432 455
1433 456
1434 457
1435 458
1436 459
1437 460
1438 461
1439 462
1440 463
1441 464
1442 465
1443 466
1444 467
1445 468
1446 469
1447 470
1448 471
1449 472
1450 473
1451 474
1452 475
1453 476
1454 477
1455 478
1456 479
1457 480
1458 481
1459 482
1460 483
1461 484
1462 485
1463 486
1464 487
1465 488
1466 489
1467 490
1468 491
1469 492
1470 493
1471 494
1472 495
1473 496
1474 497
1475 498
1476 499
1477 500
1478 501
1479 502
1480 503
1481 504
1482 505
1483 506
1484 507
1485 508
1486 509
1487 510
1488 511
1489 512
1490 513
1491 514
1492 515
1493 516
1494 517
1495 518
1496 519
1497 520
1498 521
1499 522
1500 523
1501 524
1502 525
1503 526
1504 527
1505 528
1506 529
1507 530
1508 531
1509 532
1510 533
1511 534
1512 535
1513 536
1514 537
1515 538
1516 539
1517 540
1518 541
1519 542
1520 543
1521 544
1522 545
1523 546
1524 547
1525 548
1526 549
1527 550
1528 551
1529 552
1530 553
1531 554
1532 555
1533 556
1534 557
1535 558
1536 559
1537 560
1538 561
1539 562
1540 563
1541 564
1542 565
1543 566
1544 567
1545 568
1546 569
1547 570
1548 571
1549 572
1550 573
1551 574
1552 575
1553 576
1554 577
1555 578
1556 579
1557 580
1558 581
1559 582
1560 583
1561 584
1562 585
1563 586
1564 587
1565 588
1566 589
1567 590
1568 591
1569 592
1570 593
1571 594
1572 595
1573 596
1574 597
1575 598
1576 599
1577 600
1578 601
1579 602
1580 603
1581 604
1582 605
1583 606
1584 607
1585 608
1586 609
1587 610
1588 611
1589 612
1590 613
1591 614
1592 615
1593 616
1594 617
1595 618
1596 619
1597 620
1598 621
1599 622
1600 623
1601 624
1602 625
1603 626
1604 627
1605 628
1606 629
1607 630
1608 631
1609 632
1610 633
1611 634
1612 635
1613 636
1614 637
1615 638
1616 639
1617 640
1618 641
1619 642
1620 643
1621 644
1622 645
1623 646
1624 647
1625 648
1626 649
1627 650
1628 651
1629 652
1630 653
1631 654
1632 655
1633 656
1634 657
1635 658
1636 659
1637 660
1638 661
1639 662
1640 663
1641 664
1642 665
1643 666
1644 667
1645 668
1646 669
1647 670
1648 671
1649 672
1650 673
1651 674
1652 675
1653 676
1654 677
1655 678
1656 679
1657 680
1658 681
1659 682
1660 683
1661 684
1662 685
1663 686
1664 687
1665 688
1666 689
1667 690
1668 691
1669 692
1670 693
1671 694
1672 695
1673 696
1674 697
1675 698
1676 699
1677 700
1678 701
1679 702
1680 703
1681 704
1682 705
1683 706
1684 707
1685 708
1686 709
1687 710
1688 711
1689 712
1690 713
1691 714
1692 715
1693 716
1694 717
1695 718
1696 719
1697 720
1698 721
1699 722
1700 723
1701 724
1702 725
1703 726
1704 727
1705 728
1706 729
1707 730
1708 731
1709 732
1710 733
1711 734
1712 735
1713 736
1714 737
1715 738
1716 739
1717 740
1718 741
1719 742
1720 743
1721 744
1722 745
1723 746
1724 747
1725 748
1726 749
1727 750
1728 751
1729 752
1730 753
1731 754
1732 755
1733 756
1734 757
1735 758
1736 759
1737 760
1738 761
1739 762
1740 763
1741 764
1742 765
1743 766
1744 767
1745 768
1746 769
1747 770
1748 771
1749 772
1750 773
1751 774
1752 775
1753 776
1754 777
1755 778
1756 779
1757 780
1758 781
1759 782
1760 783
1761 784
1762 785
1763 786
1764 787
1765 788
1766 789
1767 790
1768 791
1769 792
1770 793
1771 794
1772 795
1773 796
1774 797
1775 798
1776 799
1777 800
1778 801
1779 802
1780 803
1781 804
1782 805
1783 806
1784 807
1785 808
1786 809
1787 810
1788 811
1789 812
1790 813
1791 814
1792 815
1793 816
1794 817
1795 818
1796 819
1797 820
1798 821
1799 822
1800 823
1801 824
1802 825
1803 826
1804 827
1805 828
1806 829
1807 830
1808 831
1809 832
1810 833
1811 834
1812 835
1813 836
1814 837
1815 838
1816 839
1817 840
1818 841
1819 842
1820 843
1821 844
1822 845
1823 846
1824 847
1825 848
1826 849
1827 850
1828 851
1829 852
1830 853
1831 854
1832 855
1833 856
1834 857
1835 858
1836 859
1837 860
1838 861
1839 862
1840 863
1841 864
1842 865
1843 866
1844 867
1845 868
1846 869
1847 870
1848 871
1849 872
1850 873
1851 874
1852 875
1853 876
1854 877
1855 878
1856 879
1857 880
1858 881
1859 882
1860 883
1861 884
1862 885
1863 886
1864 887
1865 888
1866 889
1867 890
1868 891
1869 892
1870 893
1871 894
1872 895
1873 896
1874 897
1875 898
1876 899
1877 900
1878 901
1879 902
1880 903
1881 904
1882 905
1883 906
1884 907
1885 908
1886 909
1887 910
1888 911
1889 912
1890 913
1891 914
1892 915
1893 916
1894 917
1895 918
1896 919
1897 920
1898 921
1899 922
1900 923
1901 924
1902 925
1903 926
1904 927
1905 928
1906 929
1907 930
1908 931
1909 932
1910 933
1911 934
1912 935
1913 936
1914 937
1915 938
1916 939
1917 940
1918 941
1919 942
1920 943
1921 944
1922 945
1923 946
1924 947
1925 948
1926 949
1927 950
1928 951
1929 952
1930 953
1931 954
1932 955
1933 956
1934 957
1935 958
1936 959
1937 960
1938 961
1939 962
1940 963
1941 964
1942 965
1943 966
1944 967
1945 968
1946 969
1947 970
1948 971
1949 972
1950 973
1951 974
1952 975
1953 976
1954 977
1955 978
1956 979
1957 980
1958 981
1959 982
1960 983
1961 984
1962 985
1963 986
1964 987
1965 988
1966 989
1967 990
1968 991
1969 992
1970 993
1971 994
1972 995
1973 996
1974 997
1975 998
1976 999
1977 1000
1978 1001
1979 1002
1980 1003
1981 1004
1982 1005
1983 1006
1984 1007
1985 1008
1986 1009
1987 1010
1988 1011
1989 1012
1990 1013
1991 1014
1992 1015
1993 1016
1994 1017
1995 1018
1996 1019
1997 1020
1998 1021
1999 1022
2000 1023
2001 1024
2002 1025
2003 1026
2004 1027
2005 1028
2006 1029
2007 1030
2008 1031
2009 1032
2010 1033
2011 1034
2012 1035
2013 1036
2014 1037
2015 1038
2016 1039
2017 1040
2018 1041
2019 1042
2020 1043
2021 1044
2022 1045
2023 1046
2024 1047
2025 1048
2026 1049
2027 1050
2028 1051
2029 1052
2030 1053
2031 1054
2032 1055
2033 1056
2034 1057
2035 1058
2036 1059
2037 1060
2038 1061
2039 1062
2040 1063
2041 1064
2042 1065
2043 1066
2044 1067
2045 1068
2046 1069
2047 1070
2048 1071
2049 1072
2050 1073
2051 1074
2052 1075
2053 1076
2054 1077
2055 1078
2056 1079
2057 1080
2058 1081
2059 1082
2060 1083
2061 1084
2062 1085
2063 1086
2064 1087
2065 1088
2066 1089
2067 1090
2068 1091
2069 1092
2070 1093
2071 1094
2072 1095
2073 1096
2074 1097
2075 1098
2076 1099
2077 1100
2078 1101
2079 1102
2080 1103
2081 1104
2082 1105
2083 1106
2084 1107
2085 1108
2086 1109
2087 1110
2088 1111
2089 1112
2090 1113
2091 1114
2092 1115
2093 1116
2094 1117
2095 1118
2096 1119
2097 1120
2098 1121
2099 1122
2100 1123
2101 1124
2102 1125
2103 1126
2104 1127
2105 1128
2106 1129
2107 1130
2108 1131
2109 1132
2110 1133
2111 1134
2112 1135
2113 1136
2114 1137
2115 1138
2116 1139
2117 1140
2118 1141
2119 1142
2120 1143
2121 1144
2122 1145
2123 1146
2124 1147
2125 1148
2126 1149
2127 1150
2128 1151
2129 1152
2130 1153
2131 1154
2132 1155
2133 1156
2134 1157
2135 1158
2136 1159
2137 1160
2138 1161
2139 1162
2140 1163
2141 1164
2142 1165
2143 1166
2144 1167
2145 1168
2146 1169
2147 1170
2148 1171
2149 1172
2150 1173
2151 1174
2152 1175
2153 1176
2154 1177
2155 1178
2156 1179
2157 1180
2158 1181
2159 1182
2160 1183
2161 1184
2162 1185
2163 1186
2164 1187
2165 1188
2166 1189
2167 1190
2168 1191
2169 1192
2170 1193
2171 1194
2172 1195
2173 1196
2174 1197
2175 
```

```

1026      54         print(f"Unit at {unit_pos} healed for {heal_amount} HP!")
1027      55         Health: {self.unit_health[unit_pos]})"
1028      55         reward = 1 # Small reward for healing
1029      56     else:
1030      57         print(f"Unit at {unit_pos} is already at full health!")
1031      58         reward = -1 # Penalty for unnecessary healing
1032      59
1033      60     return reward
1034      61
1035      62 #-----
1036      63
1037      64 def player_attack(self):
1038      65     """Execute primary attack on adjacent targets"""
1039      66     reward = 0
1040      67     player_row, player_col = self.player_position
1041      68     adjacency_offsets = [(0, -1), (0, 1), (-1, 0), (1, 0)]
1042      69
1043      70     enemies_attacked = 0
1044      71     for dx, dy in adjacency_offsets:
1045      72         attack_row = player_row + dx
1046      73         attack_col = player_col + dy
1047      74         attack_pos = (attack_row, attack_col)
1048      75
1049      76         # Find enemy at this position
1050      77         for enemy in self.enemies:
1051      78             if enemy['pos'] == attack_pos:
1052      79                 damage = 25 # Player damage
1053      80                 enemy['health'] -= damage
1054      81                 enemies_attacked += 1
1055      82                 print(f"Player attacks {enemy['type']} for {damage}
1056      83                 damage! Enemy health: {enemy['health']}")"
1057      84
1058      85             if enemy['health'] <= 0:
1059      86                 print(f"{enemy['type']} defeated!")
1060      87                 reward += 10 # Reward for defeating enemy
1061      88             else:
1062      89                 reward += 2 # Small reward for successful attack
1063      90
1064      91         # Small penalty if no enemies to attack
1065      92         if enemies_attacked == 0:
1066      93             reward = -1
1067      94
1068      95     return reward
1069      96 #-----
1070      97
1071      98 def move_enemy_toward_target(self, enemy, target_pos):
1072      99     """Move enemy one step toward target"""
1073     100     reward = 0
1074     101     enemy_pos = enemy['pos']
1075     102     enemy_row, enemy_col = enemy_pos
1076     103     target_row, target_col = target_pos
1077     104
1078     105     # Calculate direction to move
1079     106     row_diff = target_row - enemy_row
1080     107     col_diff = target_col - enemy_col
1081     108
1082     109     # Choose move direction (simple pathfinding)
1083     110     move_row, move_col = 0, 0
1084     111     if abs(row_diff) > abs(col_diff):
1085     112         move_row = 1 if row_diff > 0 else -1
1086     113     else:
1087     114         move_col = 1 if col_diff > 0 else -1
1088     115
1089     116     new_row = enemy_row + move_row

```

```

1080
117     new_col = enemy_col + move_col
118
119     # Check if move is valid
120     if self._is_valid_enemy_move(enemy_pos, (new_row, new_col)):
121         self._execute_enemy_move(enemy, (new_row, new_col))
122
123     return reward
124
125 #-----
126
127 def confuse_and_teleport_enemies(self):
128     """Apply area effect that disrupts enemy positioning"""
129     reward = 0
130     # Identify all enemy positions and create a list of positions
131     enemy_positions = []
132     for row in range(len(self.map)):
133         for col in range(len(self.map[0])):
134             if self.map[row][col] in self.enemy_tiles: # Use actual enemy
135                 tiles from the map
136                 enemy_positions.append((row, col))
137     # If there are enemies on the map, confuse and possibly teleport them
138     if enemy_positions:
139         enemies_confused = 0
140         for enemy_row, enemy_col in enemy_positions:
141             # Randomly choose a direction to confuse the enemy
142             direction = random.choice(['up', 'down', 'left', 'right'])
143             teleport_possible = False
144             # Determine the new position for confusion
145             new_enemy_row, new_enemy_col = enemy_row, enemy_col
146             if direction == 'up' and enemy_row > 0:
147                 new_enemy_row -= 1
148             elif direction == 'down' and enemy_row < len(self.map) - 1:
149                 new_enemy_row += 1
150             elif direction == 'left' and enemy_col > 0:
151                 new_enemy_col -= 1
152             elif direction == 'right' and enemy_col < len(self.map[0]) - 1:
153                 new_enemy_col += 1
154             # Instead of actually moving enemies, just count confusion
155             # attempts
156             distance_to_player = abs(enemy_row - self.player_position[0])
157             + abs(enemy_col - self.player_position[1])
158             if distance_to_player <= 2: # If enemy is within close range
159                 enemies_confused += 1
160
161     # Only give reward if multiple enemies were confused
162     if enemies_confused >= 2:
163         reward = 1
164
165     return reward
166
167 #-----
168
169 def activate_and_combine_resources(self):
170     """Activates resource gathering and environmental interactionability.
171     """
172     reward = 0
173     adjacency_offsets = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
174     Left, Right
175     resource_tiles = ['R', 'F', 'W'] # R = Rock, F = Food, W = Wood
176
177     # Count adjacent resources and interactive objects
178     adjacent_resources = 0
179     adjacent_objects = 0
180
181     # Check for adjacent resources

```

```
1134 176     for dx, dy in adjacency_offsets:
1135 177         new_row = self.player_position[0] + dx
1136 178         new_col = self.player_position[1] + dy
1137 179         if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
1138 180 [0]):
1139 181             adjacent_tile = self.map[new_row][new_col]
1140 182             if adjacent_tile in resource_tiles:
1141 183                 adjacent_resources += 1
1142 184
1143 185 # Check for interactive objects nearby
1144 186 for dx, dy in adjacency_offsets:
1145 187     new_row = self.player_position[0] + dx
1146 188     new_col = self.player_position[1] + dy
1147 189     if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
1148 190 [0]):
1149 191         adjacent_tile = self.map[new_row][new_col]
1150 192         if adjacent_tile in self.interactive_object_tiles:
1151 193             adjacent_objects += 1
1152 194
1153 195 # Only give reward for significant resource/object combinations
1154 196 if adjacent_resources >= 2 and adjacent_objects >= 1:
1155 197     reward = 5 # Reward only for optimal positioning
1156 198
1157 199
1158 200
1159 201
1160 202
1161 203
1162 204
1163 205
1164 206
1165 207
1166 208
1167 209
1168 210
1169 211
1170 212
1171 213
1172 214
1173 215
1174 216
1175 217
1176 218
1177 219
1178 220
1179 221
1180 222
1181 223
1182 224
1183 225
1184 226
1185 227
1186 228
1187 229
1188 230
1189 231
1190 232
1191 233
1192 234
1193 235
1194 236
1195 237
1196 238
1197 239
1198 240
1199 241
1200 242
1201 243
1202 244
1203 245
1204 246
1205 247
1206 248
1207 249
1208 250
1209 251
1210 252
1211 253
1212 254
1213 255
1214 256
1215 257
1216 258
1217 259
1218 260
1219 261
1220 262
1221 263
1222 264
1223 265
1224 266
1225 267
1226 268
1227 269
1228 270
1229 271
1230 272
1231 273
1232 274
1233 275
1234 276
1235 277
1236 278
1237 279
1238 280
1239 281
1240 282
1241 283
1242 284
1243 285
1244 286
1245 287
1246 288
1247 289
1248 290
1249 291
1250 292
1251 293
1252 294
1253 295
1254 296
1255 297
1256 298
1257 299
1258 300
1259 301
1260 302
1261 303
1262 304
1263 305
1264 306
1265 307
1266 308
1267 309
1268 310
1269 311
1270 312
1271 313
1272 314
1273 315
1274 316
1275 317
1276 318
1277 319
1278 320
1279 321
1280 322
1281 323
1282 324
1283 325
1284 326
1285 327
1286 328
1287 329
1288 330
1289 331
1290 332
1291 333
1292 334
1293 335
1294 336
1295 337
1296 338
1297 339
1298 340
1299 341
1300 342
1301 343
1302 344
1303 345
1304 346
1305 347
1306 348
1307 349
1308 350
1309 351
1310 352
1311 353
1312 354
1313 355
1314 356
1315 357
1316 358
1317 359
1318 360
1319 361
1320 362
1321 363
1322 364
1323 365
1324 366
1325 367
1326 368
1327 369
1328 370
1329 371
1330 372
1331 373
1332 374
1333 375
1334 376
1335 377
1336 378
1337 379
1338 380
1339 381
1340 382
1341 383
1342 384
1343 385
1344 386
1345 387
1346 388
1347 389
1348 390
1349 391
1350 392
1351 393
1352 394
1353 395
1354 396
1355 397
1356 398
1357 399
1358 400
1359 401
1360 402
1361 403
1362 404
1363 405
1364 406
1365 407
1366 408
1367 409
1368 410
1369 411
1370 412
1371 413
1372 414
1373 415
1374 416
1375 417
1376 418
1377 419
1378 420
1379 421
1380 422
1381 423
1382 424
1383 425
1384 426
1385 427
1386 428
1387 429
1388 430
1389 431
1390 432
1391 433
1392 434
1393 435
1394 436
1395 437
1396 438
1397 439
1398 440
1399 441
1400 442
1401 443
1402 444
1403 445
1404 446
1405 447
1406 448
1407 449
1408 450
1409 451
1410 452
1411 453
1412 454
1413 455
1414 456
1415 457
1416 458
1417 459
1418 460
1419 461
1420 462
1421 463
1422 464
1423 465
1424 466
1425 467
1426 468
1427 469
1428 470
1429 471
1430 472
1431 473
1432 474
1433 475
1434 476
1435 477
1436 478
1437 479
1438 480
1439 481
1440 482
1441 483
1442 484
1443 485
1444 486
1445 487
1446 488
1447 489
1448 490
1449 491
1450 492
1451 493
1452 494
1453 495
1454 496
1455 497
1456 498
1457 499
1458 500
1459 501
1460 502
1461 503
1462 504
1463 505
1464 506
1465 507
1466 508
1467 509
1468 510
1469 511
1470 512
1471 513
1472 514
1473 515
1474 516
1475 517
1476 518
1477 519
1478 520
1479 521
1480 522
1481 523
1482 524
1483 525
1484 526
1485 527
1486 528
1487 529
1488 530
1489 531
1490 532
1491 533
1492 534
1493 535
1494 536
1495 537
1496 538
1497 539
1498 540
1499 541
1500 542
1501 543
1502 544
1503 545
1504 546
1505 547
1506 548
1507 549
1508 550
1509 551
1510 552
1511 553
1512 554
1513 555
1514 556
1515 557
1516 558
1517 559
1518 560
1519 561
1520 562
1521 563
1522 564
1523 565
1524 566
1525 567
1526 568
1527 569
1528 570
1529 571
1530 572
1531 573
1532 574
1533 575
1534 576
1535 577
1536 578
1537 579
1538 580
1539 581
1540 582
1541 583
1542 584
1543 585
1544 586
1545 587
1546 588
1547 589
1548 590
1549 591
1550 592
1551 593
1552 594
1553 595
1554 596
1555 597
1556 598
1557 599
1558 600
1559 601
1560 602
1561 603
1562 604
1563 605
1564 606
1565 607
1566 608
1567 609
1568 610
1569 611
1570 612
1571 613
1572 614
1573 615
1574 616
1575 617
1576 618
1577 619
1578 620
1579 621
1580 622
1581 623
1582 624
1583 625
1584 626
1585 627
1586 628
1587 629
1588 630
1589 631
1590 632
1591 633
1592 634
1593 635
1594 636
1595 637
1596 638
1597 639
1598 640
1599 641
1600 642
1601 643
1602 644
1603 645
1604 646
1605 647
1606 648
1607 649
1608 650
1609 651
1610 652
1611 653
1612 654
1613 655
1614 656
1615 657
1616 658
1617 659
1618 660
1619 661
1620 662
1621 663
1622 664
1623 665
1624 666
1625 667
1626 668
1627 669
1628 670
1629 671
1630 672
1631 673
1632 674
1633 675
1634 676
1635 677
1636 678
1637 679
1638 680
1639 681
1640 682
1641 683
1642 684
1643 685
1644 686
1645 687
1646 688
1647 689
1648 690
1649 691
1650 692
1651 693
1652 694
1653 695
1654 696
1655 697
1656 698
1657 699
1658 700
1659 701
1660 702
1661 703
1662 704
1663 705
1664 706
1665 707
1666 708
1667 709
1668 710
1669 711
1670 712
1671 713
1672 714
1673 715
1674 716
1675 717
1676 718
1677 719
1678 720
1679 721
1680 722
1681 723
1682 724
1683 725
1684 726
1685 727
1686 728
1687 729
1688 730
1689 731
1690 732
1691 733
1692 734
1693 735
1694 736
1695 737
1696 738
1697 739
1698 740
1699 741
1700 742
1701 743
1702 744
1703 745
1704 746
1705 747
1706 748
1707 749
1708 750
1709 751
1710 752
1711 753
1712 754
1713 755
1714 756
1715 757
1716 758
1717 759
1718 760
1719 761
1720 762
1721 763
1722 764
1723 765
1724 766
1725 767
1726 768
1727 769
1728 770
1729 771
1730 772
1731 773
1732 774
1733 775
1734 776
1735 777
1736 778
1737 779
1738 780
1739 781
1740 782
1741 783
1742 784
1743 785
1744 786
1745 787
1746 788
1747 789
1748 790
1749 791
1750 792
1751 793
1752 794
1753 795
1754 796
1755 797
1756 798
1757 799
1758 800
1759 801
1760 802
1761 803
1762 804
1763 805
1764 806
1765 807
1766 808
1767 809
1768 810
1769 811
1770 812
1771 813
1772 814
1773 815
1774 816
1775 817
1776 818
1777 819
1778 820
1779 821
1780 822
1781 823
1782 824
1783 825
1784 826
1785 827
1786 828
1787 829
1788 830
1789 831
1790 832
1791 833
1792 834
1793 835
1794 836
1795 837
1796 838
1797 839
1798 840
1799 841
1800 842
1801 843
1802 844
1803 845
1804 846
1805 847
1806 848
1807 849
1808 850
1809 851
1810 852
1811 853
1812 854
1813 855
1814 856
1815 857
1816 858
1817 859
1818 860
1819 861
1820 862
1821 863
1822 864
1823 865
1824 866
1825 867
1826 868
1827 869
1828 870
1829 871
1830 872
1831 873
1832 874
1833 875
1834 876
1835 877
1836 878
1837 879
1838 880
1839 881
1840 882
1841 883
1842 884
1843 885
1844 886
1845 887
1846 888
1847 889
1848 890
1849 891
1850 892
1851 893
1852 894
1853 895
1854 896
1855 897
1856 898
1857 899
1858 900
1859 901
1860 902
1861 903
1862 904
1863 905
1864 906
1865 907
1866 908
1867 909
1868 910
1869 911
1870 912
1871 913
1872 914
1873 915
1874 916
1875 917
1876 918
1877 919
1878 920
1879 921
1880 922
1881 923
1882 924
1883 925
1884 926
1885 927
1886 928
1887 929
1888 930
1889 931
1890 932
1891 933
1892 934
1893 935
1894 936
1895 937
1896 938
1897 939
1898 940
1899 941
1900 942
1901 943
1902 944
1903 945
1904 946
1905 947
1906 948
1907 949
1908 950
1909 951
1910 952
1911 953
1912 954
1913 955
1914 956
1915 957
1916 958
1917 959
1918 960
1919 961
1920 962
1921 963
1922 964
1923 965
1924 966
1925 967
1926 968
1927 969
1928 970
1929 971
1930 972
1931 973
1932 974
1933 975
1934 976
1935 977
1936 978
1937 979
1938 980
1939 981
1940 982
1941 983
1942 984
1943 985
1944 986
1945 987
1946 988
1947 989
1948 990
1949 991
1950 992
1951 993
1952 994
1953 995
1954 996
1955 997
1956 998
1957 999
1958 1000
1959 1001
1960 1002
1961 1003
1962 1004
1963 1005
1964 1006
1965 1007
1966 1008
1967 1009
1968 1010
1969 1011
1970 1012
1971 1013
1972 1014
1973 1015
1974 1016
1975 1017
1976 1018
1977 1019
1978 1020
1979 1021
1980 1022
1981 1023
1982 1024
1983 1025
1984 1026
1985 1027
1986 1028
1987 1029
1988 1030
1989 1031
1990 1032
1991 1033
1992 1034
1993 1035
1994 1036
1995 1037
1996 1038
1997 1039
1998 1040
1999 1041
2000 1042
2001 1043
2002 1044
2003 1045
2004 1046
2005 1047
2006 1048
2007 1049
2008 1050
2009 1051
2010 1052
2011 1053
2012 1054
2013 1055
2014 1056
2015 1057
2016 1058
2017 1059
2018 1060
2019 1061
2020 1062
2021 1063
2022 1064
2023 1065
2024 1066
2025 1067
2026 1068
2027 1069
2028 1070
2029 1071
2030 1072
2031 1073
2032 1074
2033 1075
2034 1076
2035 1077
2036 1078
2037 1079
2038 1080
2039 1081
2040 1082
2041 1083
2042 1084
2043 1085
2044 1086
2045 1087
2046 1088
2047 1089
2048 1090
2049 1091
2050 1092
2051 1093
2052 1094
2053 1095
2054 1096
2055 1097
2056 1098
2057 1099
2058 1100
2059 1101
2060 1102
2061 1103
2062 1104
2063 1105
2064 1106
2065 1107
2066 1108
2067 1109
2068 1110
2069 1111
2070 1112
2071 1113
2072 1114
2073 1115
2074 1116
2075 1117
2076 1118
2077 1119
2078 1110
2079 1111
2080 1112
2081 1113
2082 1114
2083 1115
2084 1116
2085 1117
2086 1118
2087 1119
2088 1110
2089 1111
2090 1112
2091 1113
2092 1114
2093 1115
2094 1116
2095 1117
2096 1118
2097 1119
2098 1110
2099 1111
2010 1112
2011 1113
2012 1114
2013 1115
2014 1116
2015 1117
2016 1118
2017 1119
2018 1110
2019 1111
2020 1112
2021 1113
2022 1114
2023 1115
2024 1116
2025 1117
2026 1118
2027 1119
2028 1110
2029 1111
2030 1112
2031 1113
2032 1114
2033 1115
2034 1116
2035 1117
2036 1118
2037 1119
2038 1110
2039 1111
2040 1112
2041 1113
2042 1114
2043 1115
2044 1116
2045 1117
2046 1118
2047 1119
2048 1110
2049 1111
2050 1112
2051 1113
2052 1114
2053 1115
2054 1116
2055 1117
2056 1118
2057 1119
2058 1110
2059 1111
2060 1112
2061 1113
2062 1114
2063 1115
2064 1116
2065 1117
2066 1118
2067 1119
2068 1110
2069 1111
2070 1112
2071 1113
2072 1114
2073 1115
2074 1116
2075 1117
2076 1118
2077 1119
2078 1110
2079 1111
2080 1112
2081 1113
2082 1114
2083 1115
2084 1116
2085 1117
2086 1118
2087 1119
2088 1110
2089 1111
2090 1112
2091 1113
2092 1114
2093 1115
2094 1116
2095 1117
2096 1118
2097 1119
2098 1110
2099 1111
2010 1112
2011 1113
2012 1114
2013 1115
2014 1116
2015 1117
2016 1118
2017 1119
2018 1110
2019 1111
2020 1112
2021 1113
2022 1114
2023 1115
2024 1116
2025 1117
2026 1118
2027 1119
2028 1110
2029 1111
2030 1112
2031 1113
2032 1114
2033 1115
2034 1116
2035 1117
2036 1118
2037 1119
2038 1110
2039 1111
2040 1112
2041 1113
2042 1114
2043 1115
2044 1116
2045 1117
2046 1118
2047 1119
2048 1110
2049 1111
2050 1112
2051 1113
2052 1114
2053 1115
2054 1116
2055 1117
2056 1118
2057 1119
2058 1110
2059 1111
2060 1112
2061 1113
2062 1114
2063 1115
2064 1116
2065 1117
2066 1118
2067 1119
2068 1110
2069 1111
2070 1112
2071 1113
2072 1114
2073 1115
2074 1116
2075 1117
2076 1118
2077 1119
2078 1110
2079 1111
2080 1112
2081 1113
2082 1114
2083 1115
2084 1116
2085 1117
2086 1118
2087 1119
2088 1110
2089 1111
2090 1112
2091 1113
2092 1114
2093 1115
2094 1116
2095 1117
2096 1118
2097 1119
2098 1110
2099 1111
2010 1112
2011 1113
2012 1114
2013 1115
2014 1116
2015 1117
2016 1118
2017 1119
2018 1110
2019 1111
2020 1112
2021 1113
2022 1114
2023 1115
2024 1116
2025 1117
2026 1118
2027 1119
2028 1110
2029 1111
2030 1112
2031 1113
2032 1114
2033 1115
2034 1116
2035 1117
2036 1118
2037 1119
2038 1110
2039 1111
2040 1112
2041 1113
2042 1114
2043 1115
2044 1116
2045 1117
2046 1118
2047 1119
2048 1110
2049 1111
2050 1112
2051 1113
2052 1114
2053 1115
2054 1116
2055 1117
2056 1118
2057 1119
2058 1110
2059 1111
2060 1112
2061 1113
2062 1114
2063 1115
2064 1116
2065 1117
2066 1118
2067 1119
2068 1110
2069 1111
2070 1112
2071 1113
2072 1114
2073 1115
2074 1116
2075 1117
2076 1118
2077 1119
2078 1110
2079 1111
2080 1112
2081 1113
2082 1114
2083 1115
2084 1116
2085 1117
2086 1118
2087 1119
2088 1110
2089 1111
2090 1112
2091 1113
2092 1114
2093 1115
2094 1116
2095 1117
2096 1118
2097 1119
2098 1110
2099 1111
2010 1112
2011 1113
2012 1114
2013 1115
2014 1116
2015 1117
2016 1118
2017 1119
2018 1110
2019 1111
2020 1112
2021 1113
2022 1114
2023 1115
2024 1116
2025 1117
2026 1118
2027 1119
2028 1110
2029 1111
2030 1112
2031 1113
2032 1114
2033 1115
2034 1116
2035 1117
2036 1118
2037 1119
2038 1110
2039 1111
2040 1112
2041 1113
2042 1114
2043 1115
2044 1116
2045 1117
2046 1118
2047 1119
2048 1110
2049 1111
2050 1112
2051 1113
2052 1114
2053 1115
2054 1116
2055 1117
2056 1118
2057 1119
2058 1110
2059 1111
2060 1112
2061 1113
2062 1114
2063 1115
2064 1116
2065 1117
2066 1118
2067 1119
2068 1110
2069
```

B.2 MECHANICS IN FIGURE 7

The following is the mechanic and helper functions for the game in Figure 7:

```
1159 1 def strategic_enemy_movement(self):
1160 2     """Process all enemy actions for this turn using A*path finding"""
1161 3     import heapq
1162 4     reward = 0
1163 5
1164 6     def heuristic(pos, goal):
1165 7         return abs(pos[0] - goal[0]) + abs(pos[1] - goal[1])
1166 8
1167 9     for enemy in self.enemies[:]:
1168 10         if enemy['finished']:
1169 11             continue
1170 12
1171 13         if self.turn_counter - enemy['last_move_turn'] >= self.
1172 enemy_move_cooldown:
1173 14             enemy['last_move_turn'] = self.turn_counter
1174 15
1175 16             # A* pathfinding to find next best move
1176 17             start = enemy['pos']
1177 18             goal = self.goal_position
1178 19
1179 20             # Priority queue: (f_score, g_score, position, path)
1180 21             open_set = [(heuristic(start, goal), 0, start, [start])]
1181 22             closed_set = set()
1182 23
1183 24             directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
1184 25             max_iterations = 50 # Limit search to prevent lag
1185 26             iterations = 0
1186 27             optimal_move = None
1187 28
1188 29             while open_set and iterations < max_iterations:
1189 30                 iterations += 1
1190 31                 f_score, g_score, current, path = heapq.heappop(open_set)
1191 32
1192 33                 if current in closed_set:
1193 34                     continue
```

```

1188 35
1189 36         if current == goal:
1190 37             # Return the next move in the optimal path
1191 38             optimal_move = path[1] if len(path) > 1 else None
1192 39             break
1193 40
1194 41             closed_set.add(current)
1195 42
1196 43             for dx, dy in directions:
1197 44                 neighbor = (current[0] + dx, current[1] + dy)
1198 45
1199 46                 if (neighbor in closed_set or
1200 47                     not self._is_valid_position_for_pathfinding(
1201 48                     neighbor)):
1202 49                     continue
1203 50
1204 51                 new_g_score = g_score + 1
1205 52                 new_f_score = new_g_score + heuristic(neighbor, goal)
1206 53                 new_path = path + [neighbor]
1207 54
1208 55                 heapq.heappush(open_set, (new_f_score, new_g_score,
1209 56                     neighbor, new_path))
1210 57
1211 58             # Execute move if valid
1212 59             if optimal_move and self._is_valid_enemy_move(enemy['pos'],
1213 60                 optimal_move):
1214 61                 self._execute_enemy_move(enemy, optimal_move)
1215 62
1216 63             # Check if enemy reached goal
1217 64             if enemy['pos'] == self.goal_position:
1218 65                 if not enemy['finished'] and not self.game_finished:
1219 66                     enemy['finished'] = True
1220 67                     self.game_finished = True
1221 68                     self.completion_order.append(f"enemy_{enemy['type']}") )
1222 69                     print(f"Enemy {enemy['type']} reached the goal first!
1223 70                     ENEMY WINS!")
1224 71                     reward -= 100 # Player loses big when enemy wins
1225 72
1226 73     return reward
1227 74
1228 75 #-----
1229 76 def _is_valid_position_for_pathfinding(self, pos):
1230 77     """Check if position is valid for pathfinding (allows temporary
1231 78     occupation)"""
1232 79     row, col = pos
1233 80     if not (0 <= row < len(self.map) and 0 <= col < len(self.map[0])):
1234 81         return False
1235 82
1236 83     tile = self.map[row][col]
1237 84     # Allow movement through walkable tiles and goal
1238 85     return tile in self.walkable_tiles or tile == 'F'
1239 86
1240 87 #-----
1241 88 def _is_valid_enemy_move(self, current_pos, new_pos):
1242 89     """Check if enemy move is valid"""
1243 90     new_row, new_col = new_pos
1244 91     if not (0 <= new_row < len(self.map) and 0 <= new_col < len(self.
1245 92         map[0])):
1246 93         return False
1247 94
1248 95     current_tile = self.map[new_row][new_col]
1249 96
1250 97     # Can move to walkable tiles or flag

```

```

1242     95     if current_tile not in self.walkable_tiles and current_tile != 'F':
1243     96         return False
1244     97
1245     98     # Cannot move to position occupied by player or other enemies
1246     99     if new_pos == self.player_position:
1247    100         return False
1248    101
1249    102     for other_enemy in self.enemies:
1250    103         if other_enemy['pos'] == new_pos:
1251    104             return False
1252    105
1253    106     return True
1254
1255
1256
1257 C QUALITY-DIVERSITY
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296

```

C.1 INITIAL MECHANICS

Here we will mention the aspects of the quality-diversity (QD) algorithm that would help in reproducibility, and were not mentioned in the main paper. The following are the initial mechanics used to initialise the QD algorithm:

```

1     mech_1 = """\n    def move_player(self, action):
2         moves = {0: (-1, 0), 1: (1, 0), 2: (0, -1), 3: (0, 1)} # Up, Down,
3         Left, Right
4         dx, dy = moves[action]
5         new_row = self.player_position[0] + dx
6         new_col = self.player_position[1] + dy
7         reward = 0
8         if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map[0]):
9             new_tile = self.map[new_row][new_col]
10            if new_tile in self.walkable_tiles:
11                self.update_player_position(new_row, new_col, new_tile)
12
13    #-----
14
15 mech_2 = """\n    def pick_object(self):
16        reward = 0
17        # Check adjacent tiles for interactive objects and pick them if
18        # present
19        adjacent_positions = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
20        Left, Right
21        for dx, dy in adjacent_positions:
22            row, col = self.player_position # player_position is in (row,
23            col) format
24            new_row = row + dx
25            new_col = col + dy
26            if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
27            [0]):
28                target_tile = self.map[new_row][new_col]
29                if target_tile in self.interactive_object_tiles:
30                    self.map[new_row][new_col] = self.default_walkable_tile
31                    reward = 1
32                    break # Exit after picking up one object
33
34    #-----
35
36 mech_3 = """\n    def hit_enemy(self):
37        reward = 0
38        # Check adjacent tiles for enemies and hit them if present

```

```

1296 37     adjacent_positions = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
1297 38     Left, Right
1298 39     for dx, dy in adjacent_positions:
1299 40         row, col = self.player_position # player_position is in (row,
1300 41         col) format
1301 42         new_row = row + dx
1302 43         new_col = col + dy
1303 44         if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
1304 45         [0]): # Check grid bounds
1305 46             target_tile = self.map[new_row][new_col]
1306 47             if target_tile in self.enemy_tiles:
1307 48                 self.map[new_row][new_col] = self.default_walkable_tile
1308 49                 reward = 1
1309 50             break # Exit after hitting one enemy
1310 51     return reward"""
1311 52 #-----
1312 53 mech_4 = """\ndef teleport_player(self):
1313 54     # Find all walkable tiles that are not adjacent to the player
1314 55     non_adjacent_walkable_positions = []
1315 56     adjacency_offsets = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
1316 57     Left, Right
1317 58     reward = 0
1318 59     # Search the map for walkable and non-adjacent tiles
1319 60     for row in range(len(self.map)):
1320 61         for col in range(len(self.map[0])):
1321 62             if self.map[row][col] in self.walkable_tiles:
1322 63                 is_adjacent = False
1323 64                 for dx, dy in adjacency_offsets:
1324 65                     if (row == self.player_position[0] + dx) and (col ==
1325 66                     self.player_position[1] + dy):
1326 67                         is_adjacent = True
1327 68                         break
1328 69                 if not is_adjacent:
1329 70                     non_adjacent_walkable_positions.append((row, col))
1330 71     # Teleport the player to a random walkable, non-adjacent position
1331 72     if non_adjacent_walkable_positions:
1332 73         new_position = random.choice(non_adjacent_walkable_positions)
1333 74         self.update_player_position(new_position[0], new_position[1],
1334 75         self.map[new_position[0]][new_position[1]])
1335 76         reward += 1
1336 77     return reward"""
1337 78 #-----
1338 79 mech_5 = """\ndef swap_positions(self):
1339 80     # Find all enemy positions on the map
1340 81     enemy_positions = []
1341 82     reward = 0
1342 83     for row in range(len(self.map)):
1343 84         for col in range(len(self.map[0])):
1344 85             if self.map[row][col] in self.enemy_tiles:
1345 86                 enemy_positions.append((row, col))
1346 87     # If there are enemies, randomly swap the player's position with an
1347 88     # enemy's position
1348 89     if enemy_positions:
1349 90         swap_with = random.choice(enemy_positions)
1350 91         enemy_row, enemy_col = swap_with
1351 92         player_row, player_col = self.player_position
1352 93         # Swap positions of player and enemy on the map
1353 94         self.map[player_row][player_col], self.map[enemy_row][enemy_col]
1354 95         = self.map[enemy_row][enemy_col], self.map[player_row][player_col]
1355 96         # Update the player's position to the swapped position
1356 97         self.player_position = (enemy_row, enemy_col)

```

```

1350 94         # Optional: Output the result of the swap
1351 95         reward += 1
1352 96     return reward"""
1353 97
1354 98 #-----
1355 99
1356 100 mech_6 = """\nndef push_object(self):
1357 101     reward = 0
1358 102     adjacent_positions = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
1359 103     Left, Right
1360 104     for dy, dx in adjacent_positions: # Swapped to dy, dx to match map
1361 105     indexing
1362 106         y, x = self.player_position # Player position is in (row, col)
1363 107         format
1364 108         new_y, new_x = y + dy, x + dx
1365 109         if 0 <= new_y < len(self.map) and 0 <= new_x < len(self.map[0]):
1366 110             # Check bounds
1367 111             target_tile = self.map[new_y][new_x]
1368 112             if target_tile in self.interactive_object_tiles:
1369 113                 push_y, push_x = new_y + dy, new_x + dx # Push in same
1370 114                 direction
1371 115                 if 0 <= push_y < len(self.map) and 0 <= push_x < len(self
1372 116                 .map[0]):
1373 117                     if self.map[push_y][push_x] in self.walkable_tiles:
1374 118                         self.map[push_y][push_x] = target_tile
1375 119                         self.map[new_y][new_x] = self.
1376 120             default_walkable_tile
1377 121                 reward = 1
1378 122             break
1379 123     return reward"""
1380 124
1381 125
1382 126 mech_7 = """\nndef jump_player(self):
1383 127     reward = 0
1384 128     # Define possible jump directions
1385 129     jump_directions = [(0, -2), (0, 2), (-2, 0), (2, 0)] # Up, Down,
1386 130     Left, Right (2 tiles)
1387 131     for dx, dy in jump_directions:
1388 132         row, col = self.player_position # player_position is in (row,
1389 133         col) format
1390 134         mid_row, mid_col = row + dx // 2, col + dy // 2 # Middle tile (
1391 135         jumped over)
1392 136         new_row, new_col = row + dx, col + dy # Landing tile
1393 137         # Check if the jump is within bounds
1394 138         if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
1395 139 [0]):
1396 140             target_tile = self.map[new_row][new_col]
1397 141             # Check if the landing tile is walkable
1398 142             if target_tile in self.walkable_tiles:
1399 143                 # Perform the jump
1400 144                 self.update_player_position(new_row, new_col, target_tile
1401 145             )
1402 146                 reward = 1
1403 147                 break # Exit after a successful jump
1404 148     return reward"""
1405 149
1406 150 mech_8 = """\nndef drop_object(self):
1407 151     reward = 0
1408 152     # Check adjacent tiles for empty walkable space
1409 153     adjacent_positions = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
1410 154     Left, Right
1411 155     for dx, dy in adjacent_positions:

```

```

1404 146     row, col = self.player_position
1405 147     new_row = row + dx
1406 148     new_col = col + dy
1407 149     # Check if position is within bounds and walkable
1408 150     if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.map
1409 151 [0]):
1410 152         if self.map[new_row][new_col] in self.walkable_tiles:
1411 153             # Place an interactive object
1412 154             self.map[new_row][new_col] = self.
1413 155             interactive_object_tiles[0] # Using first interactive object tile
1414 156             reward = 1
1415 157             break # Exit after dropping one object
1416 158     return reward"""
1417 159
1418 160 mech_9 = """\ndef enemy_move(self):
1419 161     reward = 0
1420 162     # Find all enemy positions with "#" tile on the map
1421 163     enemy_positions = []
1422 164     for row in range(len(self.map)):
1423 165         for col in range(len(self.map[0])):
1424 166             if self.map[row][col] == "#":
1425 167                 enemy_positions.append((row, col))
1426 168
1427 169     # If there are enemies, move one randomly
1428 170     if enemy_positions:
1429 171         # Pick a random enemy to move
1430 172         enemy_row, enemy_col = random.choice(enemy_positions)
1431 173
1432 174         # Define possible move directions (same as player)
1433 175         moves = {0: (-1, 0), 1: (1, 0), 2: (0, -1), 3: (0, 1)} # Up,
1434 176         Down, Left, Right
1435 177
1436 178         # Try each direction randomly until we find a valid move
1437 179         directions = list(moves.keys())
1438 180         random.shuffle(directions)
1439 181
1440 182         for action in directions:
1441 183             dx, dy = moves[action]
1442 184             new_row = enemy_row + dx
1443 185             new_col = enemy_col + dy
1444 186
1445 187             # Check if the new position is valid
1446 188             if 0 <= new_row < len(self.map) and 0 <= new_col < len(self.
1447 189 map[0]):
1448 190                 new_tile = self.map[new_row][new_col]
1449 191                 if new_tile in self.walkable_tiles:
1450 192                     # Move the enemy
1451 193                     self.map[enemy_row][enemy_col] = self.
1452 194                     default_walkable_tile
1453 195                     self.map[new_row][new_col] = "#"
1454 196                     break # Exit after successful move
1455 197
1456 198     return reward"""
1457 199
1458 mech_10 = """\ndef enemy_hit(self):
1459 200     reward = 0
1460 201     # Find all enemy positions with "#" tile on the map
1461 202     enemy_positions = []
1462 203     for row in range(len(self.map)):
1463 204         for col in range(len(self.map[0])):
1464 205             if self.map[row][col] == "#":
1465 206                 enemy_positions.append((row, col))
1466 207
1467 208     # Check if any enemy is adjacent to the player and can hit
1468 209     player_row, player_col = self.player_position
1469 210

```

```

1458 206     adjacent_positions = [(0, -1), (0, 1), (-1, 0), (1, 0)] # Up, Down,
1459 207     Left, Right
1460 207
1461 208     for enemy_row, enemy_col in enemy_positions:
1462 209         # Check if this enemy is adjacent to the player
1463 210         for dx, dy in adjacent_positions:
1464 211             check_row = enemy_row + dx
1465 212             check_col = enemy_col + dy
1466 213             # If the adjacent position matches the player's position
1467 214             if check_row == player_row and check_col == player_col:
1468 215                 # Enemy hits the player
1469 216                 reward = -1 # Negative reward for player getting hit
1470 217                 break # Exit after first hit (one enemy hitting is
1471 219             enough)
1472 220         if reward != 0: # If a hit occurred, stop checking other enemies
1473
1474
1475 C.2 GAME MECHANICS TYPES
1476
1477 We specify the types of mechanics that MORTAR uses to compute similarity scores for the Quality–Diversity archive. For each mechanic, we list its category followed by the keywords used to
1478 determine similarity.
1479

```

- **Movement:** move, walk, run, jump, fly, teleport, dash, swim, climb, crouch, sprint
- **Interaction:** pick, use, interact, open, close, talk, trade, craft, activate, push, pull
- **Combat:** attack, fight, hit, shoot, defend, block, dodge, cast, spell, heal, damage
- **Progression:** level, upgrade, unlock, improve, evolve, progress, achieve, complete, quest, mission
- **Environment:** weather, day, night, season, climate, destroy, build, terraform, grow, plant
- **Puzzle:** solve, puzzle, riddle, match, connect, arrange, decode, decipher, logic, pattern
- **Resource Management:** collect, gather, manage, inventory, store, spend, earn, balance, allocate, distribute
- **Exploration:** explore, discover, map, reveal, uncover, navigate, search, investigate, scout, survey
- **Time Manipulation:** time, slow, fast, rewind, forward, pause, resume, loop, cycle, sequence

1497 C.3 PROMPTS FOR EVOLUTIONARY OPERATORS

1498 The following are the prompts for the evolutionary operators:

1500 1. Mutation:

```

1501
1502     1     "Create a new game mechanic from the given mechanic that
1503     extends its features:\n" + solution[0] + "\n Do not make any
1504     assumptions, if you want to add a new variable or a new
1505     function, you should do it within the game mechanic method. The
1506     mechanic must return a reward, which is an integer. If a tile
1507     is being assumed then it should be defined as a single capital
1508     alphabet character and not a word. If a player is being assumed
1509     then it should be '@' tile. Remember that the game mechanic
1510     function should only take 'self' as parameter. Only output the
1511     new game mechanic as Python function, nothing else."

```

1511 2. Diversity Mutation:

```

1512      1 "Create a new game mechanic that is different, in terms of
1513      behavior of mechanics, from the ones provided:\n" + solution[0]
1514      + "\n Do not make any assumptions, if you want to add a new
1515      variable or a new funciton, you should do it within the game
1516      mechanic method. The mechanic must return a reward, which is an
1517      integer. If a tile is being assumed then it should be defined
1518      as a single capital alphabet character and not a word. If a
1519      player is being assumed then it should be '@' tile. Remember
1520      that the game mechanic function should only take 'self' as
1521      parameter. Only output the new game mechanic as Python function
1522      , nothing else."

```

3. Compatibility Mutation:

```

1524      1 "Create a new game mechanic that will make the game better
1525      when combined with the following game mechanics:\n" + solution
1526      + "\n Do not make any assumptions, if you want to add a new
1527      variable or a new funciton, you should do it within the game
1528      mechanic method. The mechanic must return a reward, which is an
1529      integer. If a tile is being assumed then it should be defined
1530      as a single capital alphabet character and not a word. If a
1531      player is being assumed then it should be '@' tile. Remember
1532      that the game mechanic function should only take 'self' as
1533      parameter. The name of the mechanic should be coherent with the
1534      behaviour of it. Only output the new game mechanic as Python
1535      function, nothing else."

```

4. Crossover:

```

1536      1 "Create a new game mechanic that combines the features of the
1537      given two mechanics to create a new game mechanic that combines
1538      the behavior of the both of them:\n" + solution + "\n Do not
1539      make any assumptions, if you want to add a new variable or a
1540      new method, you should do it within the function. The mechanic
1541      must return a reward, which is an integer. If a tile is being
1542      assumed then it should be defined as a single capital alphabet
1543      character and not a word. If a player is being assumed then it
1544      should be '@' tile. Remember that the game mechanic function
1545      should only take 'self' as parameter. The name of the mechanic
1546      should be coherent with the behaviour of it. Only output the
1547      new game mechanic as Python function, nothing else."

```

D GAMES

Play games in the user study by following the links:

1. **TreasureHunt**:<https://mortar-x3p7.onrender.com/games/TreasureHunt>
2. **HeroBreakout**:<https://mortar-x3p7.onrender.com/games/HuntBreakout>
3. **AllyCraft**:<https://mortar-x3p7.onrender.com/games/AllyCraft>
4. **CrystalCavernsCommander**:https://mortar-x3p7.onrender.com/games/Crystal_Caverns_Commander
5. **MagneticProwess**:<https://mortar-x3p7.onrender.com/games/MagneticProwess>
6. **HeroHunt**:<https://mortar-x3p7.onrender.com/games/HeroHunt>

1562
1563
1564
1565