ComplexWorld: A Large Language Model-based Interactive Fiction Learning Environment for Text-based Reinforcement Learning Agents

Anonymous Author
Affiliation
email@example.com

Abstract

Interactive fiction games have emerged as an important vehicle to improve the generalization and compositional reasoning capabilities of language-based reinforcement learning (RL) agents. Existing environments for interactive fiction games are domain-specific and do not require the RL agents to utilize compositional reasoning (sequences of interdependent decision-making capabilities to complete a task on hand). In this work, we introduce a benchmark interactive environment, *ComplexWorld*, a set of text-based games that require complex composition of previously learned skills to reach a goal. These games require the agent to understand the cause-effect relationship between the intermediary decision taken towards an overarching goal. We create and test an environment with 100 complex reasoning games, generated using an automated framework that uses large language models (GPT3) and an interactive fiction game engine (based on Inform7) to provide the user with the ability to generate more games under minimal or no human supervision. Experimental results based on both the human participants and baseline text-based RL agents reveal that current state-of-the-art text-based RL agents cannot use previously learned skills in new situations involving complex reasoning at the level humans can. These results enforce ComplexWorld's potential to serve as a sandbox environment for further research with compositional reasoning.

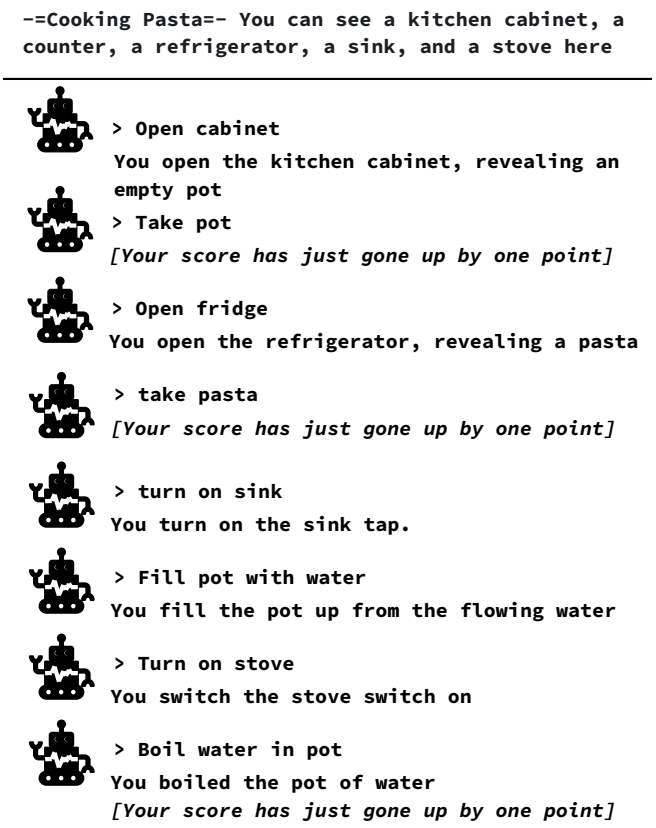

Figure 1: Example text-only agent play through of cooking pasta game. Players must use the boil skill at the correct time to be successful.

1 Introduction

Interactive fiction games such as Zork can be utilized as an important test-bed to improve the generalization and composition reasoning capabilities of text-based reinforcement learning (RL) agents [Barnes *et al.*, ; Jansen, 2022]. In these games, both the observed state of the game and the actions taken are in natural language. To play these games, the agents (or human players) need to understand the observed text from the environment and take relevant action toward the goal. These games encourage agents to utilize reasoning skills to understand the underlying state of the game. Then agents take actions to interact with the environment as depicted in Figure 1. In order to be successful, players must use previously learned skills with new objects and situations and compose these skills to complete an overarching goal. Current environments of interactive fiction games suffer from two major problems. First, environments such as TextWorld Commonsense have sacrificed game complexity for a breadth of simple reasoning games based on one-hop relationships between entities (e.g., apple → refrigerator)[Murugesan *et al.*, 2021].

This results in creating agents that do not develop or utilize complex skills involving compositional reasoning (from the environment or the external knowledge graphs). Second, environments like ScienceWorld are domain-specific so agents that play these environments may perform well while conducting specific tasks like completing science experiments but lack generalized skills to apply them to other situations [Wang *et al.*, 2022].

Therefore, the purpose of this work is to address the limitations of previous environments. First, as developing an environment of interactive fiction games is a time-intensive manual process, we develop the ComplexWorld Game Generator (CWGG) that utilizes large language models (GPT-3) and an integrated interactive fiction game engine (Inform7) to easily produce games in any domain. Second, we develop a novel text-based interactive environment using the CWGG, ComplexWorld, with 100 interactive fiction games that emphasize the need for complex compositional reasoning skills in text-based RL agents. This novel game-generation method can easily be used by users to create their own games and be adapted for future applications to build challenging RL environments in various domains. The games in ComplexWorld require agents to complete an overarching goal using a specific sequence of actions. For example, while cooking pasta, an agent must first *gather the ingredients*, *fill pot* with water, *boil the water*, and *put the pasta* in the pot. These sub-tasks are skills (such as "fill <container>", "put <object>", etc) that the agent must learn and compose in the correct order.

A growing body of work has incorporated the use of external knowledge graphs such as ConceptNet with RL and language agents to provide relevant information [Murugesan *et al.*, 2021; Kapanipathi *et al.*, 2020; Wang *et al.*, 2018]. While external knowledge has been shown to improve agent performance, we find external knowledge to have little impact on agent performance with complex reasoning tasks due to the inadequate retrieval of relevant information from one or more external knowledge graphs.

2 ComplexWorld

ComplexWorld is an interactive text-based environment that enables the text-based RL agents to hone their complex compositional reasoning skills [1]. Current environments of interactive fiction games [Murugesan *et al.*, 2021; Wang *et al.*, 2022] are often simple and domain-specific and don't require agents to utilize complex compositional reasoning skills. Environments such as Jericho [Hausknecht *et al.*, 2020] are typically not customizable and lack the ability to evaluate the text-based agent for a specific set of skills. Unlike in the other text-based environments, ComplexWorld utilizes the compositional skill generation capability of the large language models [Huang *et al.*, 2022] to generate a text-based game with a specific composition of previously learned reasoning skills. Towards this goal, we create and present the ComplexWorld Game Generator (CWGG) which takes a game idea from the user and builds a complex reasoning interactive fiction game using the GPT-3 language model [Brown *et al.*, 2020]. We design a method that procedurally generates interactive fiction games with complex reasoning using GPT-3. We initiate each game with the necessary objects that the agent needs and agents must collect those objects and use them to cook, clean, build, or complete the high-level task. In order to be successful, agents must understand the properties, location, and affordances of objects in addition to the specific sequence of actions needed to accomplish the task. Using the CWGG, we built an environment of 100 complex reasoning games for training and evaluating the text-based RL agents with compositional reasoning skills. The proposed game generator CWGG allows the users to build their own complex reasoning games with minimal supervision.

[1] In this work, we define the skills based on the actions (with intermediate reward) such as "take <object>", "fill <container>", "put <object>", etc. Compositional reasoning involves learning primitive skills and composing the previously learned skills in a correct order to achieve the final goal.

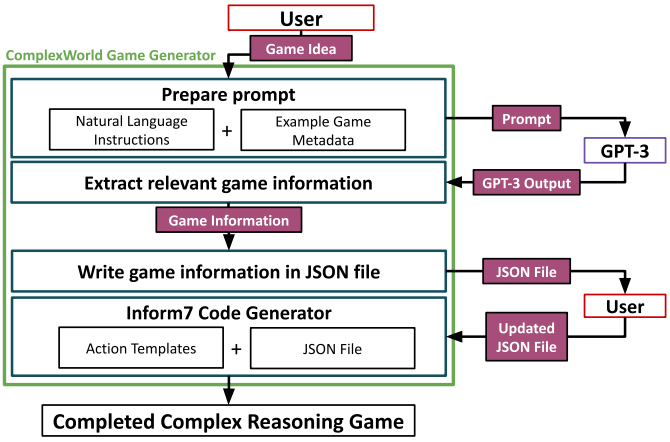

Figure 2: show the workflow of the game generator using large language model (GPT3). When the user enters a specific game idea such as "Cook pasta" into the program, the "Prepare prompt" module creates a prompt for the large language model with natural language instructions and example outputs. We feed the generated prompt to LLM of choice (GPT-3). We parse the output from LLM for the game-specific information which includes the necessary tasks, objects, and custom actions in a particular game. The information is then written into a JSON file. Optionally, at this stage, the user can either make changes to the JSON based on what they want in the game or approve the JSON file as-is. Using the JSON file, we generate an Inform7 engine-based game for a given game idea.

2.1 Constructing CWGG

Inform7 is an interactive fiction programming language that allows users to create interactive fiction games using natural language instructions [Nelson and others, 2013]. Previous text-based environments such as TextWorld, Jericho, ScienceWorld, etc use Inform7 (in the backend) to generate a handful of text-based games manually that required agents to explore a household environment and take the correct sequence of actions to complete a goal such as *cooking a pasta*. Based on our observation from these environment, we find that the game generation can be modularized into four parts: setup, object creation, custom action, and rewards sections:

1. Setup - Defined basic properties about the game such as

A

Generate a sequence of 8 actions to complete the objective. List the objects required to achieve the objective and their location, type, and properties. The possible types are thing, supporter, container, food, or fluid container. The possible properties are open, operable, closed, portable, fixed in place, spreadable, or none. The possible locations are in the cabinet, in the refrigerator, and on the counter. Then list the standard and custom actions applied to those objects. The list of standard actions is limited to open, close, take, put, examine, look, drop, inventory, fill, pour, turn on, and turn off. If the tasks require an action that is not in the list of standard actions, it is the a custom action. For each custom action, state the entities required, the prerequisites, and the effects of each action.
Example of 9 tasks for making a cake:
1. Open fridge
2. Take batter
3. Open cabinet
4. Take pan
5. Put butter in the pan
6. Turn on oven
7. Open oven
8. Put pan into the ove
9. Bake the batter with the oven

Objects required:
1. Pan; Location - in the cabinet; Type - supporter; Properties - none
2. Batter ; Location - in the refrigerator; Type - food; Properties - none

Actions applied: open, take, put, turn on
Custom Actions:
1. Bake the batter with the oven; Prerequisites - batter is in pan, pan is in oven, oven is turned on; Entities required - batter, oven; Effects - there is a cake

B

Cooking Pasta:
1. Open cabinet
2. Take pot
3. Open fridge
4. Take pot
5. Turn on sink
6. Fill pot with water
7. Turn on stove
8. Boil water in pot
9. Cook pasta with stove

Objects required:
1. Pot; location - in the cabinet; type - container; Properties - portable
2. Pasta; Location - in the fridge; Type - food; Properties - cookable

Actions applied: open, take, fill
Custom actions:
1. Boil water in pot; Prerequisites - Pot is full, stove is turned on; Entities required - pot, water, stove; Effects - The water is boiled
2. Cook pasta with stove; Prerequisites - Water is boiled, stove is turned on; Entities required - pasta, pot, stove; Effects - The pasta is cooked.

Figure 3: (A) GPT-3 input prompt for cooking games with one action example. The actual prompt contained four action examples. (B) GPT-3 output for cooking pasta game idea. GPT-3 reliably outputs accurate and necessary game information very similar to the input.

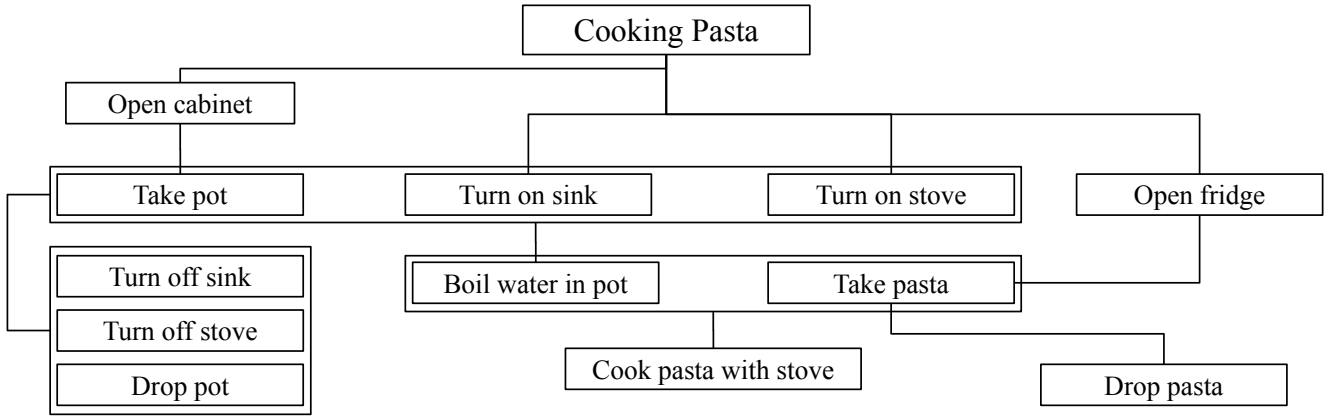

Figure 4: Composition of skills needed to complete the game "*Cooking Pasta*" as Flow diagram. A line between two skills represent that one skill needs to be executed before executing the other one (E.g., *Open cabinet* $\longrightarrow$ *Take pot*). A box with multiple skills represent that skills within the box can be executed in any order (E.g., *Boil water in pot* $\|$ *Take pasta*).

the room, any external libraries, and custom entity types.

2. Object Creation - Created in-game entities such as bread or jelly. Each entity is placed in its proper location like the refrigerator or cabinet and assigned properties such as portable, open, or closed.

3. Custom actions - Defined actions not native to Inform7. Each custom action checks for the proper pre-conditions and then executes the action by initiating the proper state changes, and reward assignments, and returning the proper observations to the agent.

4. Rewards - Assigned for gathering the necessary entities and completing custom actions to achieve the goal. Once all the rewards are collected for each game, the game ends.

To automate game generation, we split the CWGG into four modules that work chronologically to write Inform7 code for each of the four sections above. Figure 2 show an overview of CWGG. When a user feeds a game idea to the CWGG, we prepare a prompt using natural language instruction and example game metadata as shown in 3(a), with information about the setup, objects, custom actions, and rewards required for the game idea. We input this prompt to a large language model (GPT3) which outputs the requested information as shown in 3(b).

We extract the information from GPT-3 using Python simple regular expression rules by first splitting the output into three sections: tasks (ex. Open cabinet), objects (ex. Pot), and actions (ex. Fill pot with water). We add the tasks to the list of admissible actions the player could use within the game. We store the objects internally with a type, a location, a name, and a set of properties. We further split the actions

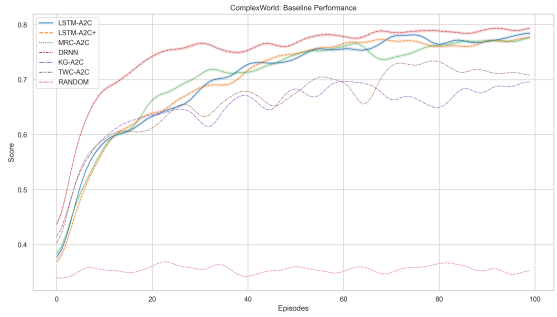

Figure 5: Number of moves

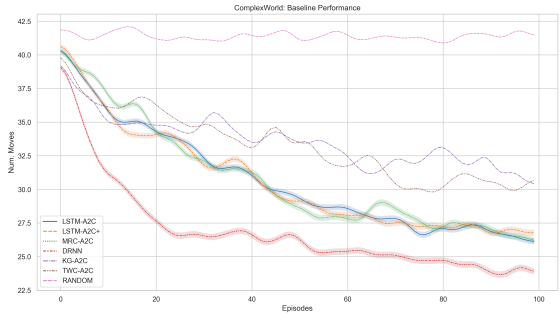

Figure 6: Normalised scores

Figure 7: Training curves depicting the scores and number of moves of text-based reinforcement learning agents.

section into default actions and custom actions which are actions native and non-native to inform7 respectively. Similar to the objects, we store each custom action internally with a name, a declaration, a definition, a set of constraints, a set of prerequisites, and a set of post-requisites.

We write the information from the GPT-3 output into a JSON file as shown in 9. The objects, actions, and tasks from the GPT-3 output correspond to the entities, custom actions, and verbs sections of the JSON file respectively. We provide an option for the user to update or change game information in the JSON file. If the user approves the game metadata in the JSON file, we write the Inform7 code and compile based on the JSON file into a completed complex reasoning game. We compile this code into the Glulx standard for interactive fiction games.

2.2 Generating Games

In order to generate games that require composing previously learned skills, we take inspiration from household chores, cooking, and maintenance tasks. We generate 100 game ideas and use the CWGG to generate an environment of 100 complex compositional reasoning games. We choose the game ideas carefully for the learning agent to utilize similar skills (ex. baking, mixing, spreading, using a hammer, etc) in new situations therefore forcing the agent to generalize skills and compose them with other skills. For example, while cooking pasta, an agent must learn how to boil water which is a skill that can be applied for a related game idea "brewing tea".

2.3 Validating ComplexWorld

We compare the complexity of ComplexWorld games with previous environments of interactive fiction games such as TextWorld Commonsense and ScienceWorld. We compare based on metrics such as the average minimum number of actions, average required rewards, and average number of skills needed to complete each game. CW games have multiple sub-tasks which indicate that agents must utilize at least 2 skills for each game in the correct order. For example, as shown in 4, one skill involves *gathering the ingredients* which must be done before the second skill of *cooking pasta* is executed. The minimum number of actions indicates that agents must take approximately 7 actions on average to complete each game, though some of these actions do not necessarily have to be completed in order (ex. the agent can "turn on stove" before "fill pot with water" and vice versa).

Games in TextWorld Commonsense only require agents to gather objects and place them in their commonsense locations. These actions can often be completed in any order whereas ComplexWorld games, such as *cooking pasta*, require agents to gather objects and use other related skills in a specific sequence.

Specifics about each game	
Min. Actions	7.36 ± 2.53
Rewards	4.08 ± 1.57
Num. Skills	2 ± 1

2.4 Human Participants

Humans are considered to have exemplary compositional reasoning skills so comparing their performance to agent performance is valuable to validate ComplexWorld's difficulty and effectiveness as a benchmark. After receiving school-level IRB approval from Pelham Memorial High School and informed consent from each of the 48 human participants, we asked the participants to play five randomly assigned complex compositional reasoning games via iplayif.com, an online interactive fiction player. Players received the goal of the game and the list of admissible actions. We collected the number of steps that each player took and the score received for each game via Google form. [2]

3 Text-based Reinforcement Learning Agents for ComplexWorld

3.1 OpenAI Gym Environment

OpenAI Gym is a general reinforcement learning framework that acts as an interface between agents and games. Gym connects environments with agents by using a monitor to keep track of every step, the final score of agents, and the sample complexity or the amount of time an agent takes to

[2] The study with high school students was IRB approved, and we obtained informed consent from all participants.

learn. OpenAI Gym was selected to track game state and agent score. Most default environments in Gym support a continuous or discrete action space although interactive fiction games require combinatorial action spaces (Hausknecht et al., 2020). TextWorld's custom Gym environment created custom data types for interactive fiction games.

3.2 Modifying TextWorld Gym for ComplexWorld

TextWorld's Gym environment only supported TextWorld-produced games that included a Glulx file and a TextWorld-generated JSON file with game metadata defined in proprietary TextWorld classes. This restricted the users ability to create games with previously undefined objects and actions which must be defined according to TextWorld's grammar and logic rules. This is a time consuming process and is prone to many errors. The goal of the ComplexWorld Game Generator is to allow users to create games without learning a new programming language or familiarizing themselves to any grammar rules, so it simulates games entirely using the Inform7 engine. This allows the user to freely define any object or action type. Therefore, ComplexWorld doesn't require meta data to track the the state of objects throughout the game. The authors wanted to use the TextWorld OpenAI Gym environment as it supports interactive fiction games. To get rid of the TextWorld json metadata dependency, an entirely new wrapper was created which acted as an interface between the CW game engine and TextWorld Gym environment. This wrapper ensures that the environment works solely with any Glulx file without the TextWorld-generated metadata. The wrapper does this by parsing the observation state returned by CW game engine after every step to generate certain data-points like admissible commands, current score, last action, number of steps taken and inventory required by the TextWorld Gym environment.

4 Experiments

4.1 Training Baseline Agents

The ComplexWorld environment includes games that require sequential reasoning. This means that, in order to successfully finish a game, an agent needs to take certain actions in a particular order. The order of actions taken decides the future states of the entities involved in the game. We trained six state of the art baseline agents on the CW game set and evaluated their ability to learn the sequential relationship between actions and objects.

4.2 RL Agents in ComplexWorld

We evaluate six baseline agents in the environment of 100 complex reasoning games. We consider text-based agents that have access to the current observation of the game and commonsense-based agents that have access to the current observation and external knowledge from ConceptNet. ConceptNet is an external knowledge graph that includes information about entities and their relationships (e.g., apple $\rightarrow$ refrigerator). To combine external knowledge with the knowledge-aware agent, we follow the methodology used by [Murugesan *et al.*, 2021]. The text-only agent, LSTM-A2C uses the currently observed text. The past observation agent, MRC-A2C uses the current and previously observed text. The commonsense agent, TWC-A2C uses the current observation and ConceptNet. The local agent, KG-A2C uses a knowledge graph of the game environment generated from the parsing of the observation states.

Data for Training and Testing The CW game set was segregated into a 75-25 train-test split. We generated 3 such train-test splits for every baseline agent.

Methodology We did three training experiments per agent on the 3 train-test splits. The agents were trained for 100 episodes on 75 games, with a batch size of 1. The maximum number of steps allowed for an agent for completing the game was 50. The training curves shown in 7 were generated after aggregating the data from 3 runs. The data in Table 1 was generated after testing the performance of trained agents on the set of 25 test games.

4.3 Results

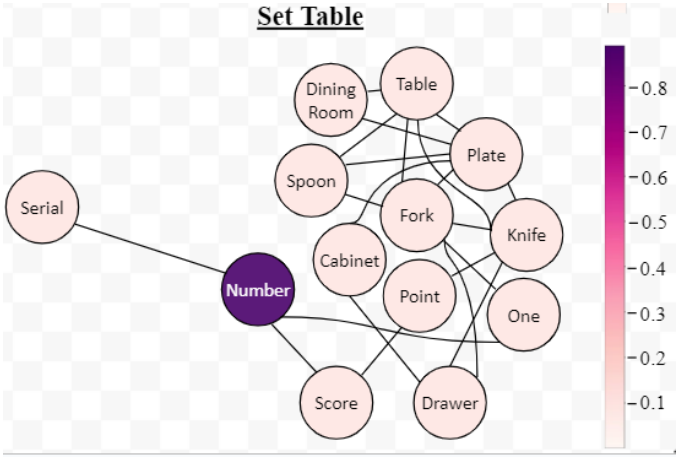

Figure 8: Knowledge graph constructed by KG-A2C agent for a game where the goal is to change set the dining table.

In the human experiments, all the participants were able to complete the games under 8-9 steps by taking the right sequence of actions. Human participants often took close to the minimum number of actions needed to accomplish each game. In Figure. 7, we can see that there was a convergence in the training curve of all the agents towards the 100th episode. The normalized scores are close to 0.75 for all of them (except the random agent which randomly selects an action). The mean number of steps taken by all the agents was almost 40. But, when it comes to solving unseen games, the agents are not even able to cross a normalized score of 0.3 with the exception of KG-A2C and MRC-A2C (by taking around 45 steps). Even the number of steps taken during training by the agents is close to 40 instead of 8-9 steps (taken by humans). This shows the reasoning complexity of the ComplexWorld and its potential of fostering the development of sequential reasoning agents. In addition, the knowledge-aware agents did not perform much better than the simple agents indicating that knowledge about the properties of entities was not always helpful to the agents. Figure 8 shows

Baseline Agent	Mean Normalized Score	Mean Moves Taken
Human	1.000 ± 0.000	9.640 ± 5.620
LSTM-A2C	0.222 ± 0.063	47.105 ± 1.876
LSTM-A2C+	0.237 ± 0.063	48.631 ± 0.886
DRRN	0.198 ± 0.065	47.035 ± 2.365
MRC-A2C	0.407 ± 0.077	43.473 ± 3.308
TWC-A2C	0.296 ± 0.079	44.122 ± 3.224
KG-A2C	0.536 ± 0.060	44.245 ± 3.001

Table 1: Performance of Baseline agents on a set of 25 unseen ComplexWorld games after training on 75 ComplexWorld games over 100 episodes.

the knowledge graph constructed by the KG-A2C agent for a sample game (Set Table) during training. This figure depicts the relationships between entities and attention weights learned by the agent for this game. Even though this learned information can be used by agent to solve similar games, perhaps, information about sequential decision making such as that found in ATOMIC may better equip agents to reason better.

5 Conclusion

We created an environment of interactive fiction games that require agents to utilize sequential decision making or complex reasoning through the modality of text. Our novel method to use the GPT-3 language model to automatically generate these games can be used to create additional complex reasoning games or adapted to build environments for new domains.

Limitations

Human participants were volunteers from Pelham Memorial High School that agreed to participate in this study. This may have introduced a bias into the human participant data since all participants were high school educated, from one geographic region, between the ages of 15 and 18, and volunteers. Many of these participants complete homework assignments and assessments often which may make their reasoning skills better than potential participants outside of school. In the future, testing human participants from various geographic locations, age groups, and levels of education may reduce bias. The CWGG currently requires human intervention to build functioning games. We will continue to work to build an end-to-end version of the CWGG, that can take a game idea and turn it into an interactive fiction game without human intervention. This would speed up development time so a larger environment of complex reasoning games can be created.

Large language models such as ChatGPT have been developed recently with the ability to interact with users in a manner conversationally similar to the interactions found in interactive fiction games. From our experimentation, ChatGPT struggles to keep track of the states of all in-game objects and the pre-conditions necessary to use those actions (ex. ChatGPT does not always require the player to turn on the stove before using it) as well as Inform7 based games. In addition, it suffers from small factual errors and is hard to reproduce the exact same result, through this could also be seen as a benefit. Despite these challenges exploring the use of models such as ChatGPT to interact with agents shows promise in the future. Pre-trained models have also not been tested with ComplexWorld which is a potential direction in the future.

References

[Barnes *et al.*,] Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. *Computer Games*, page 41.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, ..., Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[Hausknecht *et al.*, 2020] Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7903–7910, 2020.

[Huang *et al.*, 2022] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[Jansen, 2022] Peter Jansen. A systematic survey of text worlds as embodied natural language environments. In *Proceedings of the 3rd Wordplay: When Language Meets Games Workshop (Wordplay 2022)*, pages 1–15, Seattle, United States, July 2022. Association for Computational Linguistics.

[Kapanipathi *et al.*, 2020] Pavan Kapanipathi, Veronika Thost, Siva Sankalp Patel, Spencer Whitehead, Ibrahim Abdelaziz, Avinash Balakrishnan, Maria Chang, Kshitij Fadnis, Chulaka Gunasekara, Bassem Makni, Nicholas Mattei, Kartik Talamadupula, and Achille Fokoue. Infusing knowledge into the textual entailment task using

graph convolutional networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8074–8081, Apr. 2020.

[Murugesan *et al.*, 2021] Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. Text-based rl agents with commonsense knowledge: New challenges, environments and baselines. In *AAAI*, pages 9018–9027, 2021.

[Nelson and others, 2013] Graham Nelson et al. Inform 7, 2013.

[Wang *et al.*, 2018] Xiaoyan Wang, Pavan Kapanipathi, Ryan Musa, Mo Yu, Kartik Talamadupula, Ibrahim Abdelaziz, Maria Chang, Achille Fokoue, Bassem Makni, Nicholas Mattei, and Michael Witbrock. Improving natural language inference using external knowledge in the science questions domain, 2018.

[Wang *et al.*, 2022] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.

```
{"libraries" : [
   {"name": "measured liquid",
      "author": "Emily Short"},
   { "name": "modern conveniences",
      "author": "Emily Short"}],
"modules" : ["scoring"],
"room" : { "name": "home kitchen",
   "description": ""},
"custom entities" : [ "Food is a kind of thing. Food is usually edible. Food can be raw or cooked. Food is usually raw."],
"entities" : [{"name": "pot",
      "type": "container",
      "properties": ["portable", "open"],
      "location": "in the cabinet"},
   { "name": "pasta",
      "type": "food",
      "properties": "",
      "location": "in the refrigerator"},
   {"name": "sauce",
      "type": "food",
      "properties": "",
      "location": "in the refrigerator"}],
"scoring" : [
   {"condition": "taking the pot for the first time",
      "increment": "1"},
   {"condition": "taking the pasta for the first time",
      "increment": "1"},
   {"condition": "taking the sauce for the first time",
      "increment": "1"}],
"actions" : [{"name": "",
      "declaration": { "command": "cook [something] with [something]",
         "alias": "cooking it with",
         "applicable_to": "one carried thing and one thing"},
      "prerequisites": [],
      "constraints": [{"condition": "the noun is not a food",
            "prompt": "You can't cook that."},
         { "condition": "the second noun is not a stove",
            "prompt": "You can't cook that."}],
      "definition": { "tasks": [ "increase score by 1"],
         "prompt": "You cooked the [noun] with the [second noun]." },
      "postrequisties": [] } ],
"end_game" : {
   "condition": "4",
   "task": "end the story finally",
   "tasks": ["look", "inventory", "open cabinet", "take pot", “drop pot”, “open fridge”, “take pasta”, “drop pasta”, “turn on
sink”, “turn off sink”, “fill pot with water”, “turn on stove”, “turn off stove”, “boil water in pot”, “cook pasta with stove”]}}
```


Figure 9: Example JSON file produced for cooking pasta game idea. The libraries, modules, and room sections were part of the setup, the custom entities and entities sections correspond to object creation, the actions correspond to the custom actions, and the scoring and end game correspond to the rewards sections of each game. The entities section describes names, types, and properties of entities present in the game. The actions section defines custom actions including their declaration, alias, and constraints not part of Inform7 by default. The end-game section defines the maximum score and the list of admissible actions that the user can take.