COTTAGE: Coherent Text Adventure Games Generation

Yijiang River Dong

A Thesis

in

Data Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Master of Science in Engineering

2023

Supervisor of Thesis

Chris Callison-Burch

Chris Callison-Burch, Associate Professor of Computer and Information Science

Thesis Reader

Lara J. Martin, CIFellow Postdoctoral Researcher

Graduate Group Chairperson

Susan Davidson

Susan B. Davidson, Weiss Professor of Computer and Information Science

DISSERTATION TITLE

© COPYRIGHT

2023

Author Full Legal Name

This work is licensed under the
Creative Commons Attribution
NonCommercial-ShareAlike 3.0
License

To view a copy of this license, visit

http://creativecommons.org/licenses/by-nc-sa/3.0/

ACKNOWLEDGEMENT

First and foremost, I would like to express my heartfelt gratitude to my Thesis Supervisor, Professor Chris Callison-Burch, and Thesis Reader, Dr. Lara Martin, for their unwavering guidance and invaluable insights throughout my thesis work. Your exceptional mentorship, commitment to fostering a nurturing research environment, and genuine passion for innovation have not only shaped my thesis but also left an indelible mark on my personal and professional growth.

My sincerest appreciation goes out to the members of Professor Chris Callison-Burch's lab, who have generously shared their state-of-the-art research and engaged in thought-provoking discussions with me. A special mention must be made for Liam Dugan, whose infectious enthusiasm and steadfast support have been instrumental in this journey, particularly for taking the time to meticulously reading through my thesis.

I am also deeply thankful to Professor Lyle Ungar and Professor Joao Sedoc for introducing me to the fascinating field of Natural Language Processing (NLP), inspiring me to pursue a Ph.D. in NLP, and continuously providing support and mentorship.

I would like to extend my gratitude to my dear friend, Tim Shifan Dong, for his contribution to the project in its initial phase, providing me with fresh perspectives and valuable insights.

Furthermore, I am grateful to my roommates, Haotong Tian, Siming He, and Shuze Liu, as well as my friend Estelle Shen, for our engaging conversations on a wide array of topics related to artificial intelligence and deep learning. Their passion and inquisitiveness have inspired and encouraged me to pursue my research ambitions wholeheartedly.

Last but certainly not least, I owe a profound debt of gratitude to my family, whose love, support, and faith in me have been the bedrock of my success. I am who I am today because of you, and I dedicate this thesis to you with all my heart.

ABSTRACT

COTTAGE: Coherent Text Adventure Games Generation

Yijiang River Dong

Chris Callison-Burch

Text Adventure Games is a simulated text-based game environment where the player can interact via text commands and explore the story-driven environment. While traditionally Text Adventure Games are written in computer code that requires lots of effort in specifying the game, large language models like GPT-3 are shown to be able to generate interesting text adventure game

However, given the length and complexity of the game, the game generated by GPT-3 are often self-contradictory. For example, as shown in Figure 1, GPT-3 does not understand it should reject the command “shot the bodyguard” even though we specified that all bullets are exhausted.

In this thesis, we leverage neuro-symbolic reasoning to alleviate this issue. Instead of generating the text, we want to generate the code that the text adventure games run on since code is easier to be grounded than text. We first show that by one-shot prompting, GPT-4 is able to generate the code for a story-driven text adventure game. Then when the users start playing the game, we propose a system called CORRPUS (Code Representations to Reason & Prompt over for Understanding in Stories) to track the game states changes based on each GPT generation. For each GPT generation, we also check for consistencies to avoid long-range inconsistency in the generation.

Finally, we conducted human evaluations to show that the consistency and flexibility of the game all increased using our new system.

TABLE OF CONTENTS

ACKNOWLEDGEMENT ii

ABSTRACT iii

LIST OF TABLES vii

LIST OF ILLUSTRATIONS 1

CHAPTER 1 : INTRODUCTION 2

CHAPTER 2 : RELATED WORK 5

- 2.1 Text Adventure Games (TAGs) 5
- 2.2 Large Language Models (LLM) 8
- 2.3 Story Generation 10
- 2.4 Neuro-Symbolic Reasoning 11

CHAPTER 3 : CODE FORMULATION OF TEXT ADVENTURE GAME 13

- 3.1 Python Code Formulation of Text Adventure Games 13
- 3.2 Generating Static Text Adventure Games 17

CHAPTER 4 : GAME STATE TRACKING 20

- 4.1 CORRPUS State Tracking System 20
- 4.2 Adapting CORRPUS for Text Adventure Games 22

CHAPTER 5 : COTTAGE 24

- 5.1 Architecture 24
- 5.2 Plot-conditioned One-shot Static Game Generation 24
- 5.3 Retriever 26
- 5.4 Consistency Checker 26

5.5 Action Schema Generation 27
5.6 Three variants of COTTAGE 28
5.7 Example Walkthrough 29

CHAPTER 6 : EXPERIMENTS AND EVALUATION 33
6.1 Baselines 33
6.2 Evaluation Methodology 33
6.3 Results 35

CHAPTER 7 : CONCLUSION AND FUTURE WORK 40

CHAPTER 8 : Appendix 42
8.1 CoRRPUS Experiment Details 42

BIBLIOGRAPHY 47

LIST OF TABLES

TABLE 1 : An example of fine-tuning data. The full fine-tuning data is a json line file that contains hundreds of such prompt-completion pairs. . 18

TABLE 2 : An example of a few-shot prompt provided to GPT-3. The highlighted text is generated by GPT-3 as the description for `Abondoned Mine`. 19

TABLE 3 : The zero-shot prompt for GPT-3 to create a game outline and the actual outline GPT-3 generates. 25

TABLE 4 : The human evaluation results for text adventure game generation. We report the average for each metric. Recall COTTAGE-No ST is the version that does not perform state tracking and COTTAGE-AG is the version that uses action schema generation. The consistency score is the #coherent generations/#total generations. 35

TABLE 5 : Accuracy on bAbI task 2. GPT-3 with Chain-of-Thought (COT) and Selection-Inference (SI) are by Creswell et al. (2022) and GPT-3 with Dual-System (DS) is by Nye et al. (2021). The Creswell et al. (2022) models were in-house LLMs RETRO (7B parameters) (Borgeaud et al., 2021) and Gopher (280B parameters) (Rae et al., 2021) using a 5-shot prompt. When we ran their task using GPT-3 (which has 175B parameters) and a 1-shot prompt, the model did significantly better for the COT task and about the same on SI. Their numbers are reported as approximations since they were reported in a graph. All methods have been rerun on our machines. Due to pricing constraints, we ran the vanilla GPT-3 model using 1-shot prompting to more closely match our own experiments. . . . 43

TABLE 6 : ROC-AUC score on the Re^3 consistency detection task. The scores for Entailment, Entailment-DPR, and Structured-Detect models are directly cited from Yang et al. (2022). 47

LIST OF ILLUSTRATIONS

FIGURE 1 : An failure example of the text adventure game by GPT-2 powered AI Dungeon. As highlighted in red rectangles, while the player used up all the bullets, he should not be able to fire more bullets. The AI Dungeon is expected to reject the last command. 2

FIGURE 2 : A zero-shot prompting example used in Callison-Burch and Martin (2022) to describe the scenes. 9

FIGURE 3 : A few-shot prompting example used in Callison-Burch and Martin (2022) to generate descriptions of items. 9

FIGURE 4 : GPT-3's failure in tracking important game states in stories . . . 20

FIGURE 5 : An example prompt for bAbI Task 2. All three prompting methods have the same prompt world model initialization (a) followed by their respective story functions (b, c, or d), found inside the World class (end of a). That is, for the 1-shot example, CoRRPUS would be provided (a) + (b, c, or d) depending on the prompting method, and then the CODEX model is expected to output the highlighted code to update the world model. 21

FIGURE 6 : The architecture of our novel AI Dungeon COTTAGE. Given each new user command, we retrieve the relevant game history from the world model, and then conditioned on that, we prompt GPT-3 to generate the Dungeon's output to the player. Then we take the GPT-3 output into our CoRRPUS system to generate the necessary code to update the world model. Also, we note this abstraction is inspired by Figure 1 in Neural turning Machine Architecture (Graves et al., 2014) as we also believe that for today's LLMs, keeping an external memory (world model in our case) is important. 24

FIGURE 7 : All three variants of the COTTAGE systems together with two baselines that we use. The baselines will be discussed in Section 6.1 28

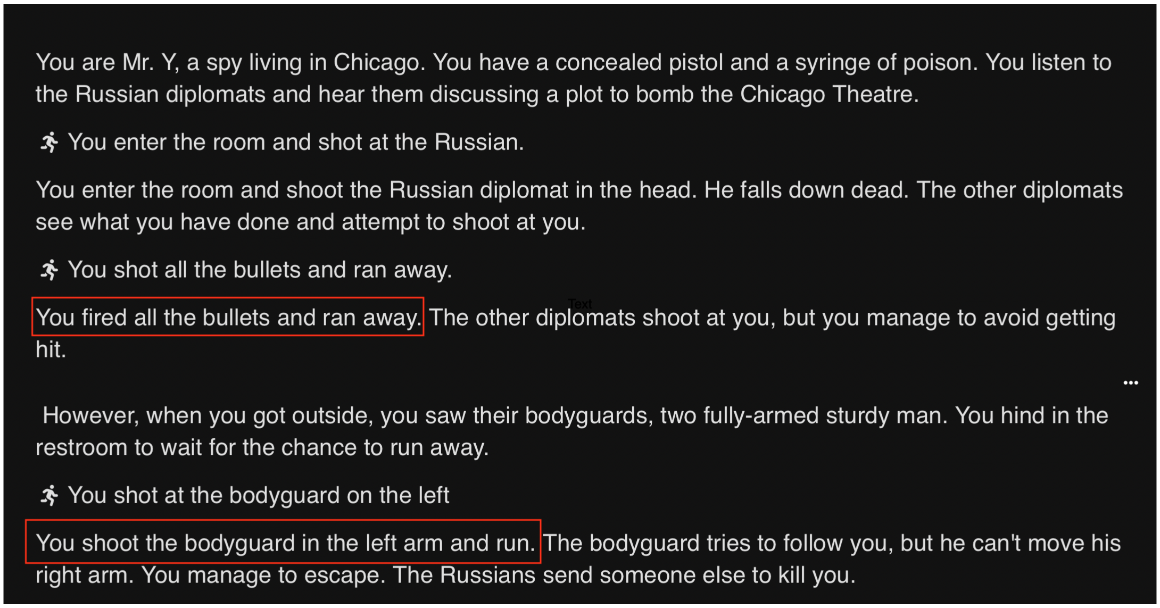
FIGURE 8 : The prompt used to ask ChatGPT to imagine itself as an AI Dungeon. 33

FIGURE 9 : Some examples of the user commands and computer responses . . 34

FIGURE 10 : An illustration of the contraction detection process of CoRRPUS for Re^3. Following Yang et al. (2022), we check attributes across the premise and the story using a BART-based entailment model and flag any contradictions for attribute-value pairs with the same attribute key. 45

CHAPTER 1 INTRODUCTION

What is a text adventure game? A text adventure game is a type of interactive fiction game where players use text-based commands to navigate through a story-driven environment. Given a goal, the player will keep performing actions (e.g., battle, use items) and the Dungeon give provide feedback accordingly until the end of the game. Readers can check out AI Dungeon[1] or Sentient Beings [2] if you are not familiar with text adventure games.



You are Mr. Y, a spy living in Chicago. You have a concealed pistol and a syringe of poison. You listen to the Russian diplomats and hear them discussing a plot to bomb the Chicago Theatre.

You enter the room and shot at the Russian.

You enter the room and shoot the Russian diplomat in the head. He falls down dead. The other diplomats see what you have done and attempt to shoot at you.

You shot all the bullets and ran away.

You fired all the bullets and ran away. The other diplomats shoot at you, but you manage to avoid getting hit.

However, when you got outside, you saw their bodyguards, two fully-armed sturdy man. You hind in the restroom to wait for the chance to run away.

You shot at the bodyguard on the left

You shoot the bodyguard in the left arm and run. The bodyguard tries to follow you, but he can't move his right arm. You manage to escape. The Russians send someone else to kill you.

Figure 1: An failure example of the text adventure game by GPT-2 powered AI Dungeon. As highlighted in red rectangles, while the player used up all the bullets, he should not be able to fire more bullets. The AI Dungeon is expected to reject the last command.

Why study text adventure games? As Jansen (2021) points out, text adventure games provide a low-cost text simulation of a 2D or 3D environment, which is an ideal testbed for embodied agents on real-world tasks. Thus, researchers can study questions including embodied reasoning, planning, Human-AI collaboration, commonsense reasoning, etc. (Jansen, 2021).

---

[1] https://aidungeon.io

[2] https://grizel.itch.io/sentient-beings

Generating text adventure games Text adventure games are traditionally controlled by the game's programming code. The programming code defines the rules of the game world and the interactions that the player can have with the environment, including the layout of the dungeon and the items and characters within it. However, the predefining all the items, characters, locations, and game rules can be labor-extensive.

With the development of Large Language Models (LLMs), researchers and entrepreneurs designed systems like AI Dungeon that could automatically generate text adventure games that can understand human input engaging and create engaging plots. However, as shown in Figure 1, these Dungeons that simply reply on LLMs can be self-contradictory mainly because it does not keep track of all the game states and does not know when how to reject impossible user requests.

Research Questions in this thesis In this thesis, I want to create an AI dungeon that can generate coherent stories based on arbitrary user inputs. To achieve this goal, the following research questions are explored in this thesis:

- **How to generate an engaging text adventure game?** In a typical human-written text adventure game, the player would travel through different locations, meet and play with different NPCs using game items. To generate a new adventure game environment, I leveraged LLM's language generation ability to initialize these locations, characters, items, etc as shown in Chapter 3 and Chapter 5. Each location/character/item would have its own descriptions and properties that could be updated as the game progresses.

- **How to maintain the long-range consistency in the generated text adventure game?** To maintain the coherence of the generated game, I incorporate neuro-symbolic reasoning and game state tracking systems as shown in Chapter 4 and Chapter 5. The game state tracking system would try to complete, update and align the game states based on the AI dungeon's text descriptions. These game states and tracking system is implemented as a code-based system to leverage the code's inherent executability and long-range reasoning nature.

Through exploring these questions, we propose our COTTAGE (**Co**heren**t** **T**ext **A**dventure **G**ames **Ge**neration) system that can generate code for a story-driven text adventure game and then act as a coherent AI dungeon for the players. We then conduct a human evaluation to show that the model is more coherent and flexible than other systems. Our code is available at `https://colab.research.google.com/drive/1Gnn6sR9oHwCHenkf3JF4ZhiiDFK7_SM0#scrollTo=0rNCXsM09Gwh`

The rest of this thesis is structured as follows

Chapter 2 provides a Literature Review on the research about text adventure games, story generation, neuro-symbolic reasoning, etc.

Chapter 3 discusses an existing code-based setup of the text adventure game that this thesis builds on. This system needs a human to pre-define all the characters, items, locations, and actions in the game.

Chapter 4 describes a novel neuro-symbolic state tracking system that we proposed and how I extend it for text adventure games.

Chapter 5 discusses how I combined the three components in Chapter 3 and Chapter 4 into an AI dungeon that can generate engaging and coherent text adventure games.

Chapter 6 presents the human evaluation that I conducted and the insight I found.

Chapter 7 concludes this thesis by discussing the limitation, implications, and future work.

CHAPTER 2 RELATED WORK

2.1. Text Adventure Games (TAGs)

History of Text Adventure Games Text adventure games have a rich history dating back to the 1960s, when computers were not easily accessible. In 1974, the first role-playing game, Dungeons & Dragons (D&D), was released and greatly influenced the future development of fantasy games. Originally used for simulating military battles, D&D quickly developed into an open-world game in which players took on different characters and performed actions (such as battles) with the help of a dungeon master, tables, maps, dice, and other tools (Callison-Burch and Martin, 2022). In comparison to role-playing games, text adventure games do not require a dungeon master and have a limited action space, making them easier for a single player to enjoy. The first text adventure game named Colossal Cave Adventure (Crowther, 1976) was created by Will Crowther in 1975. In the era before graphical user interfaces, these text-only computer games quickly gained popularity, and many more games, including Zork Lebling et al. (1979) and Planetfall, were made. Although many people shifted to graphical computer games later, the unique appeal of text adventure games has won them a great number of fans around the world, and they remain popular to this day.

Existing Text Adventure Game Environments Beyond AI Dungeon (described in the introduction), many text simulators exists nowadays for research purpose. They typically provide descriptions of the environment and the possible actions the agent can take. TextWorld (Côté et al., 2019) allows procedurally generating environment and training RL Agents. JerichoWorld (Ammanabrolu and Riedl, 2021) is a setting designed for the creation of interactive narrative games. ScienceWorld (Wang et al., 2022b) is a text adventure game that focuses on testing agents to do scientific experiments in the game. LIGHT (Urbanek et al., 2019) is more of a large-scale crowdsourced text-game resource for generating games and dialogues. However, these hard-coded game engines can only run the game, but not use AI to generate a game that is different each time based on user's input.

World Model The world models (or equivalently, environments) is the setting of our text adventure games. They are typically modeled as **Things** (locations, objects, characters) plus a set of game logic. Here we use Facebook's LIGHT (Learning in Interactive Games with Humans and Text) dataset (Urbanek et al., 2019) as an example. In LIGHT, each object has a name, id, descriptions, ex_room_ids (the ids of the room that the object is explicitly said to be located at), in_room_ids (the ids of the room that the object is not explicitly said to be located at), holding_character_ids (the ids of the character that is holding the object) as well as a set of properties such as is_food or is_gettable. Similarly, the character and room object has their corresponding fields.

Here's an example of the object "Legendary Sword":

```
{"name": "Legendary sword",
"object_id": 1188,
"descriptions": ["The sword is very old, you would assume it had once belonged to a legendary warrior.", "The sword's legend is known by everyone, it is famous throughout the land."],
"ex_room_ids": [13],
"holding_character_ids": [110],
"in_room_ids": [12],
"is_container": 0.0,
"is_drink": 0.0,
"is_food": 0.0,
"is_gettable": 1.0,
"is_plural": 0.0,
"is_surface": 0.0,
"is_weapon": 1.0,
"is_wearable": 0.0}
```

Listing 2.1: an example of LIGHT data: In the example above, base form is the general category of the object, “ex_room_ids” is the room id that the “legendary swords” is located at. “holding_character_ids” is the character id whom carries the “legendary swords”. The properties like “is_gettable” are boolean values where 1.0 indicates the object can be get by a player, vice versa.

Contemporary Research Focus: Playing Text Adventure Games This world model naturally leads to the Markov decision process (MDP) view of the text adventure game by modeling state history (S), valid state transitions (T), available actions (A), and the expected immediate reward for taking each action (R). Thus researchers propose different Reinforcement Learning (RL) algorithms to teach AI agents to play the text adventure game. Narasimhan et al. (2015) uses Deep-Q Networks and He et al. (2015) uses Deep Reinforcement Relevance Network (DRRN). Yao et al. (2020) proposes CALM that uses GPT-2 to help RL select its actions with knowledge of gameplay and commonsense. KG-A2C (Ammanabrolu and Hausknecht, 2020) shows that by using a knowledge graph to model the game states of the RL agent could lead to better performance.

In addition to RL algorithms, Wang et al. (2022a) proposes to provide the agent with Neurosymbolic modules (explained in 1.2) and then train a Decision Transformer (Chen et al., 2021a) to model the reinforcement learning as a sequence-to-sequence problem to predict the next action. However, both the RL and decision transformers require accurate modeling of the world model and extensive training. Recently, Bubeck et al. (2023) claim that GPT-4 could solve TextWorld without further training (which cannot be done by GPT-3).

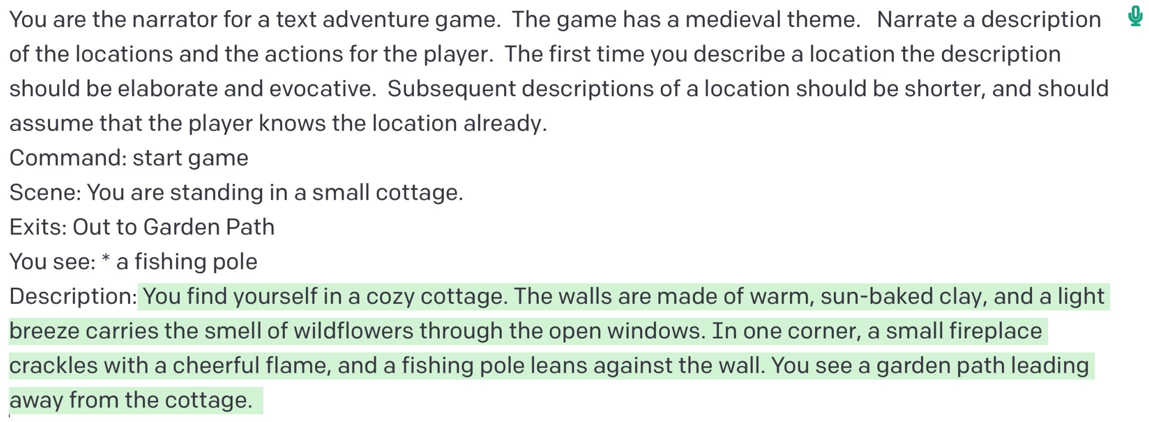
Our Research Focus: Generating Text Adventure Games It would be an important breakthrough for text adventure games and Human-AI collaboration if people could play a text adventure game that is fully generated and run by AI. Some prior work has been done to automatically generate game environments. (Fan et al., 2020) studies automatically generating coherent and creative game environments in LIGHT using BERT. Orosz (2021) fine-tunes GPT-3 on LIGHT to generate the game environment. However, they did not convert the generated environment back to games which requires iteratively updating the game states. AI Dungeon (Walton, 2019) is a GPT-based online text adventure game but it suffers from frequent inconsistencies and forgetting.

In this thesis, we attempt to create an AI Dungeon that generates and runs the text adventure game. Building on Orosz (2021), we create Python code that iteratively updates the "world model" as the game progresses and thus creates a more coherent AI Dungeon for text adventure games.

2.2. Large Language Models (LLM)

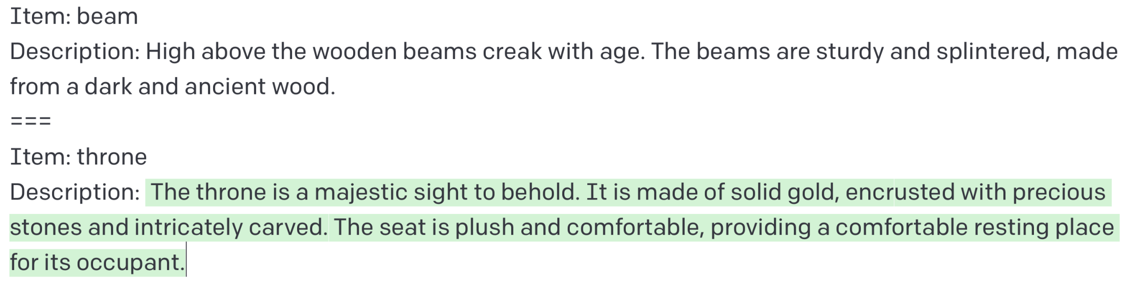
Recently by scaling language models with billions of parameters and web-scale data, large language models (LLM) achieved significant breakthroughs in various downstream tasks and unlocks many new abilities (Wei et al., 2022a). In particular, LLMs have the following notable characteristics: (1) Multi-task learners: GPT-2 (Radford et al., 2019) and T5 (Raffel et al., 2020) are multi-task learners that can solve a variety of tasks including machine translation, reading comprehension, question-and-answering, etc. Its performance can further increase when fine-tuned on task-specific data. (2) Long-form generation: GPT-3 (Brown et al., 2020) is able to generate long-form text that is nearly indistinguishable from human-written text (3) Few-shot Learning: GPT-3 can learn from in-text prompts in few-shots. (4) Instruction fine-tuning: (Ouyang et al., 2022) used Reinforcement Learning to fine-tune GPT-3 to learn from human feedback and can generate content that directly follows the prompt. (5) Multimodal: GPT-4 (OpenAI, 2023) is a multimodal LLM that aligns the embedding of text-and-image and performs well in understanding and generating images.

In particular, prompting and fine-tuning are important to the success of our COTTAGE system. We note that fine-tuning consumes more training samples and compute power and in general leads to better performance. Prompting could achieve impressive performance with the careful design of the prompts (Liu et al., 2023) although suffers from being not robust. It is generally believed that a specific, detailed prompt with several examples could lead to better performance. In our implementation, we manually compare the performance of fine-tuning and prompting and choose the better one. Below are examples of their usage.



You are the narrator for a text adventure game. The game has a medieval theme. Narrate a description of the locations and the actions for the player. The first time you describe a location the description should be elaborate and evocative. Subsequent descriptions of a location should be shorter, and should assume that the player knows the location already.
Command: start game
Scene: You are standing in a small cottage.
Exits: Out to Garden Path
You see: * a fishing pole
Description: You find yourself in a cozy cottage. The walls are made of warm, sun-baked clay, and a light breeze carries the smell of wildflowers through the open windows. In one corner, a small fireplace crackles with a cheerful flame, and a fishing pole leans against the wall. You see a garden path leading away from the cottage.

Figure 2: A zero-shot prompting example used in Callison-Burch and Martin (2022) to describe the scenes.



Item: beam
Description: High above the wooden beams creak with age. The beams are sturdy and splintered, made from a dark and ancient wood.
===
Item: throne
Description: The throne is a majestic sight to behold. It is made of solid gold, encrusted with precious stones and intricately carved. The seat is plush and comfortable, providing a comfortable resting place for its occupant.

Figure 3: A few-shot prompting example used in Callison-Burch and Martin (2022) to generate descriptions of items.

Despite these impressive capacities of LLMs, many major issues still remain unsolved, including hallucination, long-range inconsistency, AI safety, and multi-hop reasoning. Given the goal of generating coherent text adventure games, the issue of long-range inconsistency is particularly concerning.

In story generation, many methods have been proposed to alleviate this issue. Fan et al. (2018) proposes to generate multiple candidate sentences and then rerank them based on their coherence score. Yang et al. (2022); Nye et al. (2021) propose to maintain a database of world states to eliminate contradictory generations. Building on these works, we propose CoRRPUS (Code Representations to Reason & Prompt over for Understanding in Stories) (Dong et al., 2022a) to leverage LLM's impressive programming capacities to do neuro-symbolic reasoning to track the world states in the story (described in Section 4.1). In our implementation, we adopt CoRRPUS to track game states in text adventure games. Then, similar to Yang et al. (2022) and Nye et al. (2021), given the extracted world model, we check the consistency of between the generation and the world model to alleviate the issue of long-range inconsistency.

2.3. Story Generation

In this section, we discuss automatic storytelling as it shares many similarities with creating a TAG world.

An important line of research is about planning the plots of stories. Rashkin et al. (2020) proposes to generate multi-paragraph stories conditioned on a story outline using transformers with memory attention. Yao et al. (2019) used schema to plan out the entire storyline and then use seq-2-seq models to generate the story. Martin et al. (2018) explored using Recurrent Neural Network (RNN) to do event generation and thus the full story. Tambwekar et al. (2018) aims at controlling the story plot by controlling its ending and events order via reinforcement learning. Yang et al. (2022) recursively prompt GPT-3 to generate the characters, and story outlines to write coherent stories of thousands of words long that go beyond GPT-3's context length. In this thesis, we let the user specify the high-level goal, and then by one-shot prompting, we let GPT-3 generate the game with a plot.

Other researches such as character-centric generation are also relevant to text adventure games. Character-centric generation studies how different characters would respond to the same actions differently and talk differently. Character-centric generation could improve the way NPCs interact with the player in the game (Louis and Sutton, 2018) (Ammanabrolu et al., 2020). In this thesis, we generate text conditioned on the characters (which contains his persona, characteristics, etc.) and the surrounding environment which enables the model to take this information into account.

2.4. Neuro-Symbolic Reasoning

Before the rise of deep learning or connectionism, symbols are viewed as the root of intelligent action (Newel and Simon, 1976). Although deep learning dominates the field of artificial intelligence in the past decade and achieve human-level performance on many tasks, it has many apparent weaknesses including interpretability, robustness, efficiency, and generalization. Therefore, many researchers consider combining these structured symbolic methods with probabilistic neural networks has grown. As Dr. Yoshua Bengio drew from Nobel Price Winning Psychologist Daniel Kahneman's book "Thinking Fast and Slow" (Kahneman, 2011) to argue that artificial intelligence needs to combine system 1 and system 2 thinking like human, where system 1 represents the intuitive and fast thinking like a neural network and system 2 represents the slow, logical, and reasoning-related thinking (Bengio, 2019). This comment highlights the biological motivation for neuro-symbolic reasoning.

Post LLMs Since the rise of LLMs, researchers found that LLMs can learn to perform symbol manipulations by few-shot examples or fine-tuning. Chen et al. (2021b) published CODEX by fine-tuning GPT-3 on GitHub code and shows LLM's capacity to do program synthesis. Schick et al. (2023) proposes toolformer, which teaches LLMs when and how to APIs during decoding. Similar ideas are carried out by OpenAI to create ChatGPT Plugins, through which GPT can learn to use hundreds of APIs provided by third-party developers. In particular, the WolframAlpha plugin in particular equips GPT with many powerful symbolic modules that would in part solve GPT's math shortcomings as a language model (Wolfram, 2023). Jiang et al. (2022) and Wu et al. (2022) show that LLM can be used to assist theorem proving. Surís et al. (2023) proposes ViperGPT that teaches LLMs to perform interpretable neuro-symbolic reasoning across vision and language. With their world knowledge and commonsense, LLMs can also directly write machine code to plan and control robots (Huang et al., 2022) (Liang et al., 2022).

Neuro-Symbolic Reasoning in Story Generation As discussed in Section 2.3, many researchers plan out the story plot and then generate the full story. Other researches include structure to enhance story generation including using external resources. For example, (Huang et al., 2020) uses ConceptNet (Speer et al., 2017) (Martin, 2021) uses VerbNet (Brown et al., 2019), (Peng et al., 2021) uses ATOMIC (Sap et al., 2019).

The recent work on the extraction of a story world or world model is particularly relevant to this thesis. These use LLMs to dynamically maintain an external "world model" representation to keep track of the story and maintain the story consistency. (Martin, 2021) (Yang et al., 2022) uses a python dictionary, (Nye et al., 2021) uses object-oriented notation, and (Andrus et al., 2022) uses knowledge graphs.

Neuro-Symbolic Reasoning in This Thesis As traditional text adventure games are built on predefined computer code (symbolic), it is natural to consider the role of symbolic modules in the automatic generation of text adventure games. There are two key observations on why and how to use symbolic modules in generating text adventure games. (1) Similar to Nye et al. (2021), Yang et al. (2022), Dong et al. (2022b), and Figure 1, we note that LLMs often generate self-contradictory text especially when prompted to do so in GPT-powered AI Dungeon. Thus we provide examples and APIs for the code to iteratively update the "world model" and check for consistency when a new action is an input by the player. (2) Different from story generation, text adventure games require the system to have a much deeper understanding of each action: its preconditions and effects and act according to the "world model". In this thesis, we let LLMs to generate the action schema to reject invalid actions and update the game states.

CHAPTER 3 CODE FORMULATION OF TEXT ADVENTURE GAME

In this chapter, we introduce two text adventure game systems that this thesis builds on. One system is the python environment on which we can build hard-corded text adventure games. The other component is about fine-tuning LLMs to generate a *static* text adventure game.

3.1. Python Code Formulation of Text Adventure Games

Things Following the LIGHT data, one crucial component of Text Adventure Games are the locations, items, and characters, which we call `Things`. All these things have their own descriptions and properties that we should keep track in our "world model". We thus create the following `Thing` super class with class `Item, Character` or `Location` that inherit from it.

```
class Thing:
  def __init__(self, name, description):
    self.name = name
    self.description = description
    # A dictionary of properties and their values. Boolean properties for items
        include: is_gettable, is_wearable, is_drink, is_food, is_weapon, etc.
    self.properties = defaultdict(bool)
```

Listing 3.1: Thing class

```
class Item(Thing):
  def __init__(self, name, description, examine_text, take_text="", start_at=None):
    super().__init__(name, description)
    self.examine_text = examine_text
    self.take_text = take_text if take_text else ("You take the %s." % self.name)
    # If an item is gettable, then the player can get it and put it in their
        inventory.
    self.properties["gettable"] = True
    self.commands = {}
    self.location = start_at
```

Listing 3.2: Item class (The commands will be explained in the next paragraph. It relates to how we can perform actions using the item.)

Actions In addition to initializing the `Things`, the code formulation code also needs to define common and special action functions that allow the players to actually interact with the environment. While many different formulations are possible, we implement the actions in python as an objects as shown in 3.3. The Action fields contain preconditions and effects and when the player performs the action, the code will check the preconditions. If all conditions are met, the action will be performed and the effects will also be triggered. Otherwise, the action will be rejected by the Dungeon.

```
class Action:
  """
  Action schema have a set of preconditons and effects that result from taking the
      action, if the preconditons.
  """
  def __init__(self, parameter_types, preconditons, effects):
    # preconditions = list of tuples (test_function [args = list of param names])
    self.preconditons = preconditons
    self.effects = effects

  def check_preconditions(self, parameters):
    # parameters = map from parameter_name to param itself
    for precondition, arguments in self.preconditons:
      if not precondition(parameters, arguments):
        return False
    return True

  def try_action(self, **kwargs):
    parameters = kwargs
    if self.check_preconditions(parameters):
      for effect, arguments in self.effects:
        effect(parameters, arguments)
      return True
    else:
      return False
```

Listing 3.3: Action Class (simplied version)

For preconditions, each contains a list of tuples that define the conditions that must be satisfied for the action to be executed. Each tuple contains a test function and a list of parameter names that will be passed as arguments to the function. The test function should return a Boolean value. If all test functions return ‘True’, the action can be executed. Similarly, effects are lists of tuples which is the function and their arguments. These effect functions will be executed by passing the arguments in if the preconditions are met.

With these preconditions and effects defined, we can then define some common actions such as `Go`, `Get`, `Drop`, etc. Listing 3.4 is an example of Go. The users can also define their own special functions as shown in Listing 3.5.

```
preconditons = [
    (at, ("character", "from_loc")),
    (connected, ("from_loc", "to_loc", "direction")),
    (not_blocked, ("from_loc", "direction")) ]
```

```
effects = [
    (move, ("character", "from_loc", "to_loc", "direction"))
]

go = Action(parameter_types, preconditons, effects)
```

Listing 3.4: The implementation of Go Action Schema

```
class Item(Thing):
  def __init__:
    ...

  def add_action(self, command_text, function, arguments, preconditions={}):
    """Add a special action associated with this item"""
    self.commands[command_text] = (function, arguments, preconditions)

  def do_action(self, command_text, game):
    """Perform a special action associated with this item"""
    end_game = False  # Switches to True if this action ends the game.
    if command_text in self.commands:
      function, arguments, preconditions = self.commands[command_text]
      all_conditions_met, failure_reasons = check_preconditions(preconditions, game)
      if all_conditions_met:
        end_game = function(game, arguments)
    else:
      gpt_describe(failure_reasons, [], f"You cannot perform {command_text}")
    return end_game

## Then, assume we have initialized an object rosebush, we can add a special function
    to it by writing the code below.
rosebush.add_action("pick rose",  add_item_to_inventory, (rose, "You pick the lone
    rose from the rosebush.", "You already picked the rose."), preconditions={"
    inventory_not_contains": rose}
```

Listing 3.5: The implementation of do_action that allows we perform a special function to an item. And an example of special function "pick rose" that one can perform to item rose. As we can see, the "command text" is pick rose. The precondition is that inventory does not contains rose and the effect is to add rose into the inventory.

Game In this way users can create a code-based text adventure game by specifying the environment and actions. The game typically ends with a special function `end_game` that indicates if the play has won or lost the game. Listing 3.6 is an example of the game and Listing 3.7 is the transcript when playing the game.

```python
def build_game():
  # Locations
  cottage = Location("Cottage", "You are standing in a small cottage.")
  garden_path = Location("Garden Path", "You are standing on a lush garden path.
      There is a cottage here.")
  fishing_pond = Location("Fishing Pond", "You are at the edge of a small fishing
      pond.")

  # Connections
  cottage.add_connection("out", garden_path)
  garden_path.add_connection("south", fishing_pond)

  # Items that you can pick up
  fishing_pole = Item("pole", "a fishing pole", "A SIMPLE FISHING POLE.", start_at=
      cottage)
  rose = Item("rose", "a red rose", "IT SMELLS GOOD.",  start_at=None)
  fish = Item("fish", "a dead fish", "IT SMELLS TERRIBLE.", start_at=None)

  # Sceneary (not things that you can pick up)
  pond = Item("pond", "a small fishing pond", "THERE ARE FISH IN THE POND.", start_at
      =fishing_pond)
  pond.set_property("gettable", False)
  rosebush = Item("rosebush", "a rosebush", "THE ROSEBUSH CONTAINS A SINGLE RED ROSE.
        IT IS BEAUTIFUL.", start_at=garden_path)
  rosebush.set_property("gettable", False)

  # Add special functions to your items
  rosebush.add_action("pick rose",  add_item_to_inventory, (rose, "You pick the lone
      rose from the rosebush.", "You already picked the rose."))
  pond.add_action("catch fish",  describe_something, ("You reach into the pond and
      try to catch a fish with your hands, but they are too fast."))
  pond.add_action("catch fish with pole",  add_item_to_inventory, (fish, "You dip
      your hook into the pond and catch a fish.","You weren’t able to catch another
      fish."), preconditions={"inventory_contains":fishing_pole})
  fish.add_action("eat fish",  end_game, ("That’s disgusting! It’s raw! And
      definitely not sashimi-grade! But you’ve won this version of the game. THE END.
      "))

  locations = [fishing_pond, garden_path, fishing_pond]
  items = [fishing_pole, rose, fish, pond, rosebush]
  characters = []

  player = Character(name="player", description="You are a simple peasant destined
      for greatness.", persona="I am on an adventure.", location=cottage)

  return Game(cottage, player, items, characters, locations)
```

Listing 3.6: An example of the full game in Python

Play The user could start playing the game by entering the command. The parser class would first parse the command to get the intention of the user, then call the corresponding functions to check the preconditions and trigger the effects. Finally, conditioned on the command and recent game states, the system calls GPT-3 to provide users a vivid description of what happened after the command. An example of user playing the game is shown in Listing 3.7.

```
You are standing in a small cottage.
Exits:
Out to Garden Path

You see:
 * a fishing pole
 * a poisonous potion

---

You find yourself standing in a small cottage. The walls are made of roughly
hewn logs, and the roof is covered with thick thatch. The wooden floor creaks as
you step inside, and the smell of freshly baked bread fills the air. There is a
small wooden table in the corner of the room, and next to it sits a fishing
pole. On the table lies a small vial of what appears to be a poisonous potion.
Out the door you can see a winding garden path, lit by the evening sun. What do
you do?

Player: take pole

You grab the fishing pole off the table. Its handle is cool to the touch, and
the line looks freshly waxed. What do you do now?

Player: go out
Player is at the Garden Path.

Player: go south
Player is at the Fishing Pond.

Player: catch fish with pole

You cast your line into the shimmering waters of the pond. After a few minutes,
you feel a tug on your line and reel in a silvery fish. You can now add the fish
to your inventory. What do you do now?
```

Listing 3.7: An example transcript that when playing the text adventure game in 3.6. “Player” is the player’s input command and all other texts are generated by GPT-3.

3.2. Generating Static Text Adventure Games

This thesis also builds on Orosz (2021) and HW2 of Callison-Burch and Martin (2022)’s course, which shows that GPT-3 is able to generate a *static* text adventure game via fine-tuning or few-shot prompting. By static, we note that the model only generates the descriptions and properties of the Things but does not create an AI Dungeon that the player can interact with.

Specifically, Orosz (2021) experiment with fine-tuning and few-shot prompting LLMs for predicting the descriptions and properties of Things. In 1 and 2, we show an example of the fine-tuning data and few-shot examples as well as the corresponding model output that they used.

{“prompt”: “The rectory”, “completion”: “This room is quite small and cramped. It’s about the size of maybe three wooden carts, which is to say, it’s very small. There are boxes all over the place and many candles and other church accessories. There are several big robes hanging next to what looks like a very small closet. Some candles shed an eerie light on the room, flickering softly. There is a small cabinet with several religious tapes and records and a few books. A book case is near and contains many common religious texts.”}

Table 1: An example of fine-tuning data. The full fine-tuning data is a json line file that contains hundreds of such prompt-completion pairs.

Then, for generating each of the Things via fine-tuning and few-shot prompting, Orosz (2021) conducted a detailed human evaluation and concluded that the performance of fine-tuning with GPT-3 Curie is better than few-shot prompting GPT-3 Curie and is on par with few-shot prompting GPT-3 Davinci. (Recall that GPT-3 has four versions: Ada, Babbage, Curie, and Davinci. Davinci is the most powerful and expansive one and Curie is cheaper but not as powerful as Davinci. Ada and Babbage are less powerful than Curie.) And thus it ies not worth the extra cost to use GPT-3 Davinci to conduct the experiments. This thesis follows the same setup: we mainly use GPT-3 Curie and prefer fine-tuning when there exists sufficient data.

In summary, this chapter discusses (1) the code-based representation that our COTTAGE system builds on and (2) a method to generate a static game. But (1) requires the user to hard-code all the things and game logic in the game while (2) is only a static game that users cannot play. The next chapter introduces a novel state-tracking system that is crucial for us to maintain long-range consistency in the final COTTAGE system.

Location: Cottage safe-room
Description: Stocked from floor to ceiling, food and medicine line the western wall. The room is painted matte grey and contains a chest to the left of the food and medicine. Three chairs and a table with a lamp are what is left in the middle of the safe-room. Dimly lit, and a steady water supply coming from the well outside of the cottage.

===

Location: Cottage entryway
Description: A welcome mat adorns the floor at the foot of the door. A rack to hang hats and coats is on the left hand side. Right above, a fan lazy spins about on a hot summer day. A single bulb of light illuminates the small hall that the entrance leads to. The entrance smells of the fresh outdoors.

===

Location: Unexplored jungle
Description: Located outside of civilization, the jungle is a vast unknown. Many plants have never seen, many beasts never discovered. However, what is known is that one cannot survive alone in this jungle. There are many poisonous trees and plants and it is impossible to see the sun once in it. Just an incredible amount of vast, dense and impassable vegetation.

===

Location: Abandoned Mine
Description: Deep in the mountains, the abandoned mine lies forgotten by most. The entrance is covered in vines and cobwebs, and a deep, thick silence hangs heavy in the air. Once inside, the darkness of the mine is illuminated by a faint, eerie light, coming from the distant depths of the mine. Here and there, rusty tools and broken pieces of machinery can be seen, and the walls are lined with large ore veins. Despite its long abandonment, the mine still holds its secrets, waiting to be discovered.

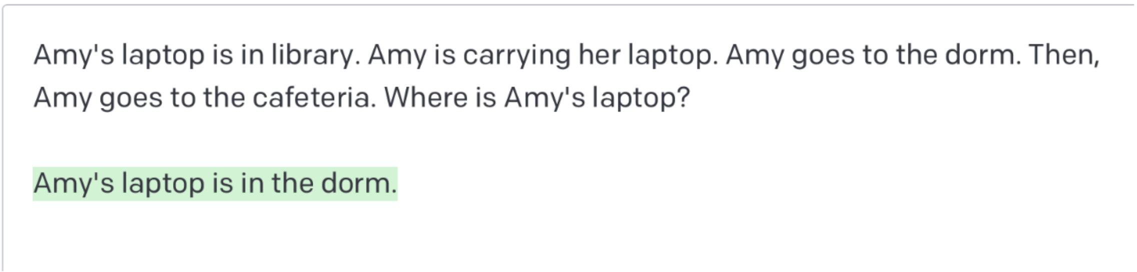
Table 2: An example of a few-shot prompt provided to GPT-3. The highlighted text is generated by GPT-3 as the description for `Abondoned Mine`.

CHAPTER 4 GAME STATE TRACKING

To solve the issue of generating inconsistent text in text adventure games, we want to keep track of the "world model" via an external knowledge representation. This chapter introduces the neuro-symbolic state tracking system that we use. We first introduce our previous work (Dong et al., 2022a) CORRPUS, which is a state tracking system for stories, and then show how we adopt it for text adventure games.

4.1. CORRPUS State Tracking System

In Dong et al. (2022b), We proposed CORRPUS (**Co**de **R**epresentations to **R**eason & **P**rompt over for **U**nderstanding in **S**tories) to use neuro-symbolic reasoning to improve story understanding. The motivation is similar to Figure 1, for stories, LLMs also sometimes lose track of the important game states 4 and generate self-contradictory stories. Thus we proposed a novel CODEX-based neuro-symbolic state tracking system. The idea is that given a world model W_0 that contains all the characteristics that we want and the story S, we convert them into code representation and ask CODEX to generate code to update the world model.



Amy's laptop is in library. Amy is carrying her laptop. Amy goes to the dorm. Then, Amy goes to the cafeteria. Where is Amy's laptop?

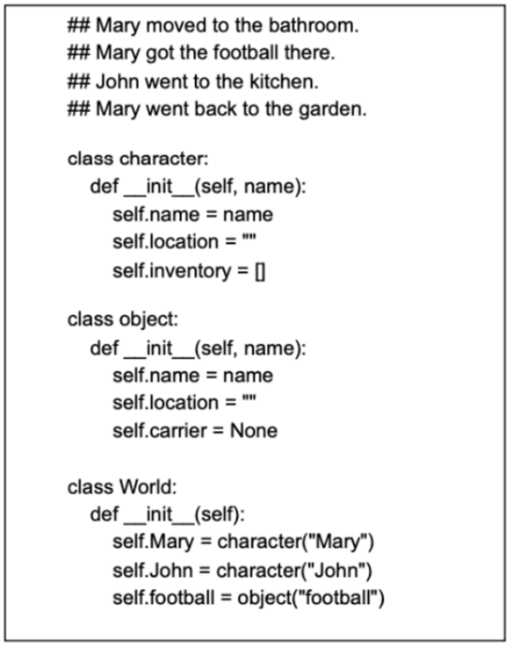
Amy's laptop is in the dorm.

Figure 4: GPT-3's failure in tracking important game states in stories

Formally, we present the problem as follows: given a story $\mathcal{S} = [S_0 : S_1 : ... : S_n]$ and an initial state of the story world $\mathcal{W}_0$, the goal is to have the model update the story world state $\mathcal{W}_i$ until the story reaches its conclusion, $\mathcal{W}_n$. After each sentence $\mathcal{S}_i$ within the story, an unspecified update function $\mathcal{U}_i$ is used by the model to modify the world state. Consequently, the final world state $\mathcal{W}_n$ is derived through the following process:

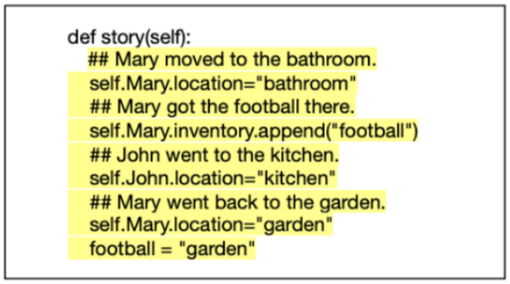
$\mathcal{W}_0 \xrightarrow{\mathcal{U}_0} \mathcal{W}_1 \xrightarrow{\mathcal{U}_1} .... \xrightarrow{\mathcal{U}_{n-1}} \mathcal{W}_{n-1} \xrightarrow{\mathcal{U}_n} \mathcal{W}_n$. It is important to note that, apart from $\mathcal{W}_0$ and $\mathcal{W}_n$, the intermediate world states remain hidden and are not visible to the user.

CoRRPUS, the system we have developed, extracts structured data using code-based chain-of-thought-type prompting, allowing for more precise tracking of the underlying story state and identification of inconsistencies.



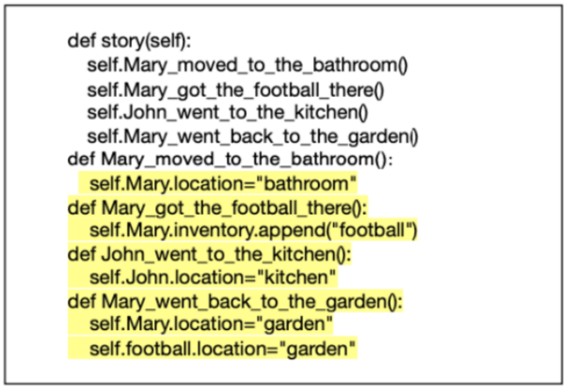
```
## Mary moved to the bathroom.
## Mary got the football there.
## John went to the kitchen.
## Mary went back to the garden.

class character:
  def __init__(self, name):
    self.name = name
    self.location = ""
    self.inventory = []

class object:
  def __init__(self, name):
    self.name = name
    self.location = ""
    self.carrier = None

class World:
  def __init__(self):
    self.Mary = character("Mary")
    self.John = character("John")
    self.football = object("football")
```

(a) Prompt initialization



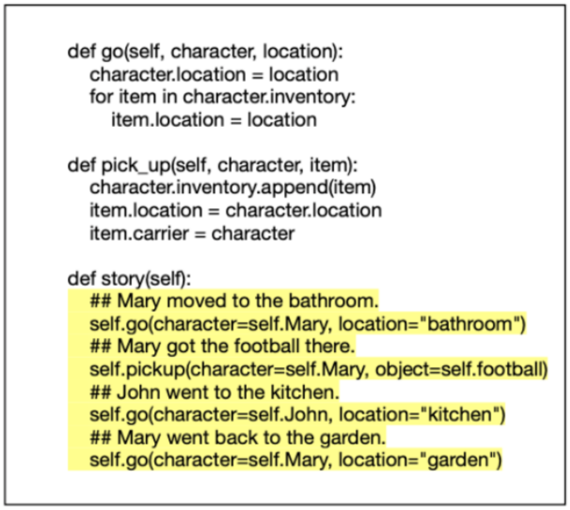
```
def story(self):
  ## Mary moved to the bathroom.
  self.Mary.location="bathroom"
  ## Mary got the football there.
  self.Mary.inventory.append("football")
  ## John went to the kitchen.
  self.John.location="kitchen"
  ## Mary went back to the garden.
  self.Mary.location="garden"
  football = "garden"
```

(b) Prompting with Comments



```
def story(self):
  self.Mary_moved_to_the_bathroom()
  self.Mary_got_the_football_there()
  self.John_went_to_the_kitchen()
  self.Mary_went_back_to_the_garden()
def Mary_moved_to_the_bathroom():
  self.Mary.location="bathroom"
def Mary_got_the_football_there():
  self.Mary.inventory.append("football")
def John_went_to_the_kitchen():
  self.John.location="kitchen"
def Mary_went_back_to_the_garden():
  self.Mary.location="garden"
  self.football.location="garden"
```

(c) Prompting with Specific Functions



```
def go(self, character, location):
  character.location = location
  for item in character.inventory:
    item.location = location

def pick_up(self, character, item):
  character.inventory.append(item)
  item.location = character.location
  item.carrier = character

def story(self):
  ## Mary moved to the bathroom.
  self.go(character=self.Mary, location="bathroom")
  ## Mary got the football there.
  self.pickup(character=self.Mary, object=self.football)
  ## John went to the kitchen.
  self.go(character=self.John, location="kitchen")
  ## Mary went back to the garden.
  self.go(character=self.Mary, location="garden")
```

(d) Prompting with Abstract Functions

Figure 5: An example prompt for bAbI Task 2. All three prompting methods have the same prompt world model initialization (a) followed by their respective story functions (b, c, or d), found inside the World class (end of a). That is, for the 1-shot example, CoRRPUS would be provided (a) + (b, c, or d) depending on the prompting method, and then the CODEX model is expected to output the highlighted code to update the world model.

We experiment with three different prompting techniques as shown in 5. The following examples demonstrate how the story sentence “Sandra journeyed to the bedroom” would be altered for each type of prompt.

- **Comment Only**: This is the most basic prompt, offered no additional structural information. It must rely solely on the comments from the prompt and initialization of the story world state. This reveals the ability of the Code-LLM to deduce and complete information with minimal guidance. Example: Given the comment `## Sandra journeyed to the bedroom.`, the model should output `self.Sandra.location = "bedroom"`.

- **Specific Functions**: This prompt transforms each sentence into a distinct function. Alongside the commented sentences at the beginning of the prompt, the system is also prompted to create functions for each sentence. Example: Given the function name `Sandra_journeyed_to_the_bedroom()`, the model should produce a definition for the function, which should include `self.Sandra.location = "bedroom"`.

- **Abstract Functions**: This final prompting style provides functions for actions or setting attributes to the model. The model does not need to determine the meaning of a specific event but must identify the appropriate function for a given story sentence and how to fill in the function’s arguments. Example: Given `go(character, location)` and other functions, generate `go(character=Sandra, destination=bedroom)`.

We conduct two set of experiments on bAbI Task 2 and Re^3 and showed that CORRPUS greatly improves the tracking system. Details can be found in Appendix (Section 8.1).

4.2. Adapting CORRPUS for Text Adventure Games

To use CoRRPUS for text adventure games, we first add two functions (update_description and update_properties) to Things to update its descriptions and properties. Then, we retrieve the relevant game states and ask GPT-3.5 (ChatGPT) to update the world model as CoRRPUS. The relevant game states now is all the items and characters at the location and all the items in the player’s inventory. An example is shown in Listing 4.1.

```
Game states:
1. Location:
Dark Room: You are in a dark room that you can barely see anything in the room.
2. Existing Items:
pond: a small fishing pond
lamp: a lamp (unlit)
pole: a fishing pole
3. Existing Characters:
player: You are a simple peasant destined for greatness.
---
Action: Lit up the lamp
---
Description: You lit up the candle and  the room was suddenly filled with light. You
    can now see the pond, the fishing pole, and a few other items in the room.
---
Now write code to update the game states only when necessary.
Code:
## Add missing items/characters that is missing in the Game states.
## No changes needed
## Update existing item/character's properties and descriptions
game.items['lamp'].update_description("A lamp (lit)")
game.items['lamp'].update_properties("Lit", True)
```

Listing 4.1: The prompt we use for CoRRPUS on text aventure games. We show GPT the game states (location, items, and characters), the user input, and GPT generated descriptions, then we ask GPT to generate the code to update the world model. We also note that since CODEX API is no longer available, we used GPT3.5-turbo for the CoRRPUS on text adventure games based on OpenAI's discovery that GPT3.5 is equally good at CODEX in terms of coding-related tasks.

CHAPTER 5 COTTAGE

5.1. Architecture

Recall the main goal of this thesis is to alleviate the issue of long-range consistency by keeping track of the world model. In the previous two sections, we introduced the static game generator and state tracking system (CoRRPUS). In this section, we put these components together to construct our COTTAGE system as shown in 7.

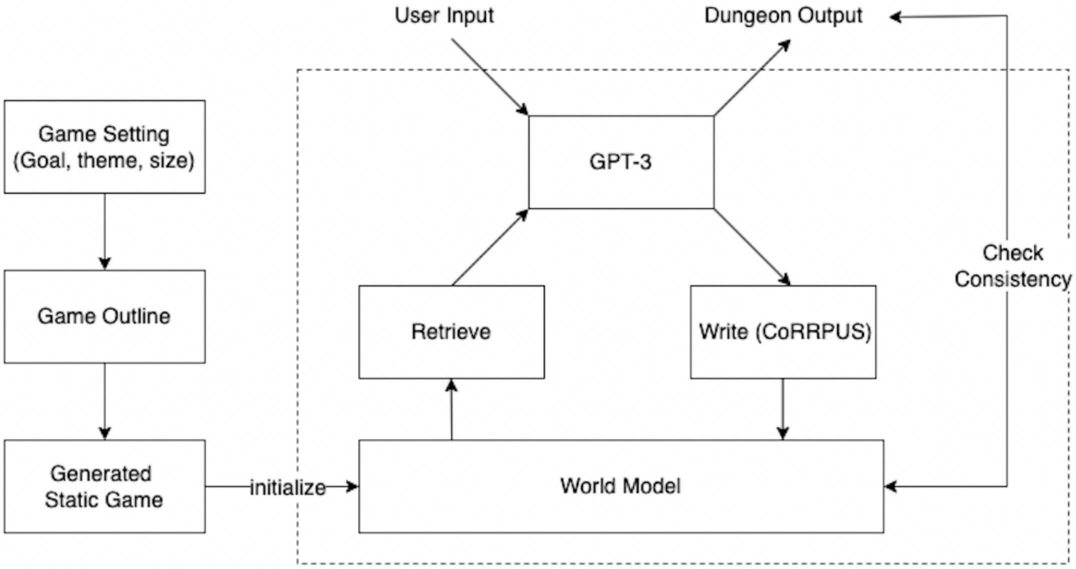


Figure 6: The architecture of our novel AI Dungeon COTTAGE. Given each new user command, we retrieve the relevant game history from the world model, and then conditioned on that, we prompt GPT-3 to generate the Dungeon's output to the player. Then we take the GPT-3 output into our CoRRPUS system to generate the necessary code to update the world model. Also, we note this abstraction is inspired by Figure 1 in Neural turning Machine Architecture (Graves et al., 2014) as we also believe that for today's LLMs, keeping an external memory (world model in our case) is important.

5.2. Plot-conditioned One-shot Static Game Generation

While Orosz (2021) studies generating descriptions and properties of Things one by one, it does not consider how to compose them in a meaningful way to form a text adventure game. While we also experimented with hierarchically generating each description and then putting them together, we found that the most effective way is to use what we call "plot-conditioned One-shot Generation". We first prompt GPT-3 with the goal (or win condition) of the game as well as the size of the game (e.g., the expected number of rooms, characters, and items) and then ask it to outline the locations, characters, and items in the game. The prompt and outline are shown in 3. Then, we take the prompt in 3, the generated outline, and an example game from 3.6, and ask GPT-4 to write code to expand the outline to a static text adventure game. The generated game is shown in 6.4. Note that since the code generation step can easily go beyond the context length of GPT-3, we opt to use GPT-4 for generating the code.

Now let's generate a text adventure game. There should be rooms, characters, and items in the game. The goal is to defeat Lord Voldemort. You should generate rooms, characters, items following the goal.
Room 1: Hogwarts Entrance Hall
Characters: Harry Potter, Dumbledore, Professor McGonagall, Lord Voldemort Items: Wand, Cloak of Invisibility, Philosopher's Stone
Room 2: Great Hall
Characters: Ron Weasley, Hermione Granger, Hagrid
Items: Triwizard Cup, Gryffindor Sword, Marauder's Map
Room 3: Forbidden Forest
Characters: Aragog, Centaur, Dragon
Items: Golden Snitch, Time Turner, Basilisk Fang
Room 4: Chamber of Secrets
Characters: Tom Riddle, Salazar Slytherin, Basilisk
Items: Sorting Hat, Sword of Gryffindor, Crucio Curse Book

Table 3: The zero-shot prompt for GPT-3 to create a game outline and the actual outline GPT-3 generates.

```
def build_game():
  # Locations
  hogwarts_entrance_hall = Location("Hogwarts Entrance Hall", "You are standing in
      the Hogwarts Entrance Hall.")
  ...

  # Items that you can pick up
  wand = Item("wand", "a wand", "A MAGIC WAND.", start_at=hogwarts_entrance_hall)
  ...

  # Characters
  harry_potter = Character("Harry Potter", "Harry Potter is a young wizard with a
      lightning-shaped scar.", "Hello, I'm Harry Potter.", location=
      hogwarts_entrance_hall)
  ...
```

```
# Actions for defeating Lord Voldemort
voldemort.add_action("defeat voldemort", destroy_item, (voldemort, "Using your wand, you cast a
    powerful spell and defeat Lord Voldemort. Congratulations! You win the game!", "You have
    already defeated Lord Voldemort."), preconditions={"inventory_contains": wand,
    "inventory_contains": sword_of_gryffindor}


player = Character(name="player", description="You are a brave wizard at Hogwarts,
    ready to save the school from evil.", persona="I am on a mission to defeat Lord
     Voldemort.", location=hogwarts_entrance_hall)
locations = [hogwarts_entrance_hall, great_hall, forbidden_forest,
    chamber_of_secrets]
characters = [harry_potter, dumbledore, mcgonagall, voldemort, ron_weasley,
    hermione_granger, hagrid, aragog, centaur, dragon, tom_riddle,
    salazar_slytherin, basilisk]
items = [wand, cloak, philosophers_stone, triwizard_cup, gryffindor_sword,
    marauders_map, golden_snitch, time_turner, basilisk_fang, sorting_hat,
    sword_of_gryffindor, crucio_curse_book]

game = Game(hogwarts_entrance_hall, player, items, characters, locations)
return game
```

Listing 5.1: The game generated by GPT-4. It's directly executable building on the code described in Chapter 3. In particular, this game is carefully designed toward the goal of defeating voldemort. As I highlight the special action "defeat voldmort" that is generated by GPT-4, the preconditions and effects are highly non-trival and coherent with the game setting.

5.3. Retriever

To limit the input length for GPT-3, we retrieve the most relevant game states and feed that into GPT-3. We retrieve the name and descriptions of all the items and characters in the player's location.

5.4. Consistency Checker

After the static game is generated, the users can start playing the game. The way we prompt GPT-3 to create Dungeon output is the same as Section 3.1. However, sometimes the Dungeon output can be self-contradictory as shown in Figure 1. Thus we need to check if the Dungeon output is consistent with the world model. If the output is consistent, we use CoRRPUS to update the game state. If it's inconsistent, the output is regenerated. We go over an example to help illustrate the mechanics of the the pipeline in 5.7. This section will focus on how to check the consistency between the Dungeon output and the game states.

In practice, we prompt GPT-3 to determine if the description and the retrieved game states are consistent.

5.5. Action Schema Generation

In the COTTAGE standard system discussed in 7, the player are limited to perform the predefined special functions. To make the system more flexible, we would want the system to be able to take in arbitrary commands and if it is a new action, our system generate the action schema for it. This is achieved via “Action Schema Generation”.

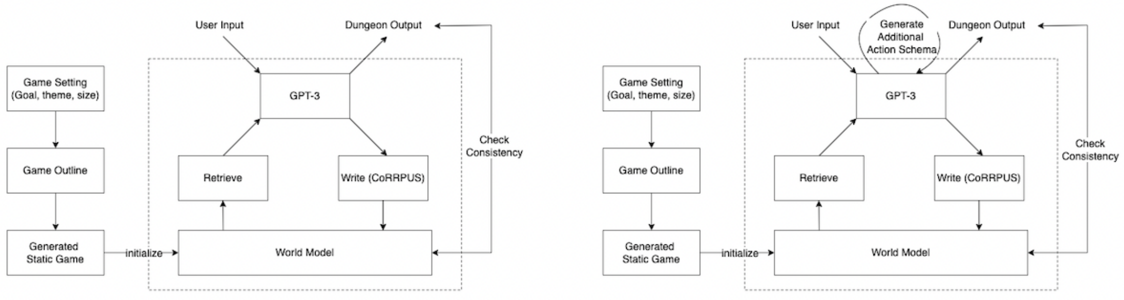
```
## Omitted the definition of the Item Class and predefined precondition and effect
    functions

## Now generate a new action schema for "pick rose" using the defined preconditions
    and effects. You should carefully consider the precondition and effects of each
    action. (Only use at and inventory_contains for precondition.)
game.items = {'red rose': <__main__.Item at 0x7fb2888c97b3>}
game.characers = {}

## Action Schema:
## the relavant object is red rose
try:
  game.items['red rose'].add_action("pick rose",  add_item_to_inventory, (game.items[
      'red rose'],"You pick the lone rose from the rosebush.","You already picked the
      rose."),  preconditions={})
  self.run_special_command("pick rose")
except:
  print("error in generated action schema")
===
## Now generate a new action schema for "catch fish with pole" using the defined
    preconditions and effects. You should carefully consider the precondition and
    effects of each action. (Only use at and inventory_contains for precondition.)

game.items = ‘fishing pole’: <__main__.Item at 0x7fb43765c97b3>, ‘fish’: <__main__.Item at
    0x7fe32765c97b3>
game.characers = {}

## Action Schema:
## the relevant object in the game states is fishing pole
try:
 game.items['fishing pole'].add_action("catch fish with pole", add_item_to_inventory,
    (game.items['fish'], "You dip your hook into the pond and catch a fish.","You weren't able to
    catch another fish."), preconditions={"inventory_contains": game.items['fishing pole']})
 self.run_special_command("catch fish with pole")
except:
 print("error in generated action schema")
```

Listing 5.2: The example prompt that we use for action schema generation. The text highlighted in pink are those that are retrieved and changed for different settings. The function highlighted in yellow is the action schema generated. It successfully writes executable code and writes the correct effect. It also stated the precondition that fishing

pole should be in our inventory.

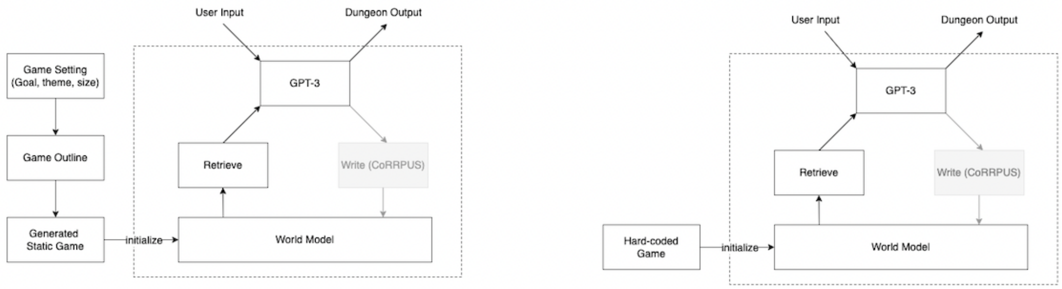
5.6. Three variants of COTTAGE

By combing the components described above, we finally have the following three variants of COTTAGE systems. They are graphically illustrated in 7.



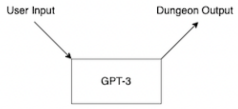
(a) COTTAGE-Standard

(d) COTTAGE-AG



(b) COTTAGE-No ST

(e) Hard-coded Game



(c) ChatGPT's self imagination

Figure 7: All three variants of the COTTAGE systems together with two baselines that we use. The baselines will be discussed in Section 6.1

(1) COTTAGE Standard: This is our base system. It combines all the crucial components including game generation, CoRRPUS state tracking, consistency checker, and GPT-3 generation.

(2) COTTAGE Without State Tracking (COTTAGE-No ST): In this variant, we disable the world model updating step (CoRRPUS) to examine how much the state tracking system

affects the coherence of the game

(3) COTTAGE With Action Schema Generation (COTTAGE-AG): In this variant, we add the action schema generation module to examine how much the level of flexibility it adds and whether or not it causes inconsistent generations. Assuming the action schema generation works well, we would have **a variant of COTTAGE that can take in arbitrary user input** and generate consistent Dungeon output.

5.7. Example Walkthrough

Below we go over an example of the COTTAGE Standard to help the reader better understand how the whole pipeline works. We continue with the Harry Potter example in which the static game is generated in Listing 3 and Listing 6.4.

A typical game loop starts with the user entering a command. If the command is successfully parsed by our parser, the AI dungeon would output a description based on the action. The text highlighted in pink is the underlying game state representation that is not shown to the player. The longer descriptions highlighted in yellow are actual text that will be shown to the players. Then the consistency checker will check the generation and if it's consistent, the CoRRPUS state tracker will update the game states accordingly. Then the COTTAGE system would wait the player to enter the new command.

Finally, the player successfully defeat Voldemort (with the wand and sword) and win the game.

```
> Typically here the player should enter a new command but the player does not do it for the first
    round.
---
You are standing in the Hogwarts Entrance Hall.
    Exits:
    North to Great Hall
    You see:
    * a wand
    * a cloak of invisibility
    * a philosopher"s stone

---

You enter the grand and foreboding Hogwarts Entrance Hall. A chill runs downyour spine as you take in
    the sight of the ancient stone walls and toweringwindows. The air is thick with the dust of years
    and a sense of magic hangs inthe air. You see a wand, a cloak of invisibility, and a
    philosopher"s stone laidout on a nearby table. North to Great Hall: You walk through the
    archwayleading to the Great Hall. This majestic room is filled with ornate tapestriesand banners
    of ancient houses. At the end of the hall is a large fireplace, anda long table that could easily
    seat hundreds of students.
---
Consistency Checker: CONSISTENT
---

update_game_state:
## Add missing items/characters that is missing in the Game states.
## No changes needed
## Update existing item/character"s properties and descriptions
game.locations["Hogwarts Entrance Hall"].update_description("You enter the grand and
    foreboding Hogwarts Entrance Hall. A chill runs down your spine as you take in
    the sight of the ancient stone walls and towering windows. The air is thick with
    the dust of years and a sense of magic hangs in the air. You see a wand, a cloak
    of invisibility, and a philosopher"s stone laid out on a nearby table.")
game.locations["Great Hall"].update_description("This majestic room is filled with
    ornate tapestries and banners of ancient houses. At the end of the hall is a
    large fireplace, and a long table that could easily seat hundreds of students.")
game.locations["Hogwarts Entrance Hall"].add_exit("north", game.locations["Great Hall
    "])

>take wand

---
 You take the wand.
---

You take the wand from the table and feel a surge of power course through your
body. You look around the entrance hall, taking in the sights and feeling the
magic in the air.
---
Consistency Checker: CONSISTENT
---

update_game_state:
## Add missing items/characters that is missing in the Game states.
## No changes needed
## Update existing item/character"s properties and descriptions
game.items["wand"].update_description("A powerful wand that feels like an extension
    of your arm.")
game.characters["player"].update_properties("Has wand", True)

>go north
Player is at the Hogwarts Entrance Hall.
trying to go north to the Great Hall.
Player is at the Great Hall.

---
 This majestic room is filled with ornate tapestries and banners of ancient houses.
     At the end of the hall is a large fireplace, and a long table that could easily
     seat hundreds of students.
Exits:
South to Hogwarts Entrance Hall
East to Forbidden Forest

You see:
 * a triwizard cup
 * a gryffindor sword
 * a marauder"s map

---

You stand in the Great Hall, a majestic room with ornate tapestries and banners
of ancient houses. At the end of the hall sits a large fireplace, and a long
```

```
table that could easily seat hundreds of students. In the center of the room,
you spot a Triwizard Cup, a Gryffindor Sword and a Marauder"s Map.
---
Consistency Checker: CONSISTENT
---

update_game_state:
## No changes needed.

>take gryffindor sword

---
 You take the gryffindor sword.
---

You are still in the Great Hall, the same ornate room from before. You now carry
the Gryffindor Sword, which you snatched from the center of the room.
---
Consistency Checker: CONSISTENT
---

update_game_state:
## Add missing items/characters that is missing in the Game states.
## No changes needed
## Update existing item/character"s properties and descriptions
game.items["gryffindor sword"].update_description("A powerful sword that glows with a
     red light")
game.items["gryffindor sword"].update_properties("Carried", True)
game.characters["player"].update_description("You are a brave wizard at Hogwarts,
    ready to save the school from evil. You now carry the Gryffindor Sword.")
game.characters["player"].update_properties("Carrying Sword", True)

>go south
Player is at the Great Hall.
trying to go south to the Hogwarts Entrance Hall.
Player is at the Hogwarts Entrance Hall.

---
 You enter the grand and foreboding Hogwarts Entrance Hall. A chill runs down your
     spine as you take in the sight of the ancient stone walls and towering windows.
     The air is thick with the dust of years and a sense of magic hangs in the air.
     You see a wand, a cloak of invisibility, and a philosopher"s stone laid out on a
      nearby table.
Exits:
North to Great Hall

You see:
 * a cloak of invisibility
 * a philosopher"s stone

---

You are now in the Hogwarts Entrance Hall, a grand and foreboding room. The
ancient stone walls and towering windows make you shiver. The wand, cloak of
invisibility and philosopher"s stone still lay on the nearby table. You can go
north to the Great Hall.

---
Consistency Checker: CONSISTENT
---

update_game_state:
## Add missing items/characters that is missing in the Game states.
```

```
gryffindor_sword = Item("gryffindor sword", "sword", "A powerful sword that glows
    with a red light", start_at=player)
game.add_item(gryffindor_sword)

## Update existing item/character"s properties and descriptions
game.characters["player"].update_description("You are a brave wizard at Hogwarts,
    ready to save the school from evil. You now carry the Gryffindor Sword.")
game.locations["Hogwarts Entrance Hall"].update_description("You enter the grand and
    foreboding Hogwarts Entrance Hall. A chill runs down your spine as you take in
    the sight of the ancient stone walls and towering windows. The air is thick with
    the dust of years and a sense of magic hangs in the air. You see a wand, a cloak
    of invisibility, and a philosopher"s stone laid out on a nearby table.")
game.items["wand"].update_description("A powerful wand that feels like an extension
    of your arm.")
game.items["cloak"].update_description("A cloak of invisibility")
game.items["philosopher\"s stone"].update_description("A philosopher"s stone")
game.locations["Hogwarts Entrance Hall"].update_exit("north", "Great Hall")

>defeat voldemort

---
 defeat voldemort
---

You draw my wand and point it at Lord Voldemort. You then reach for the Gryffindor
    sword, feeling its weight and power in my hands. You begin to cast powerful
    spells, and the sword acts as an extension of your wand, amplifying the power of
    upir spells and deflecting Voldemort"s curses. You press forward, pushing him
    back until he is weakened and defeated. With a final, powerful stroke from your
    sword, Lord Voldemort is vanquished. You win the game!

---
Consistency Checker: CONSISTENT
---

update_game_state:
## Add missing items/characters that is missing in the Game states.
great_hall = Location("Great Hall", "The Great Hall is a grand room with high
    ceilings and long tables for the Hogwarts students.")
game.add_location(great_hall)

## Update existing item/character"s properties and descriptions
game.items["gryffindor sword"].update_properties("Power", 10)
game.characters["Harry Potter"].update_description("Harry Potter is a young wizard
    with a lightning-shaped scar. He is currently studying at Hogwarts.")
game.characters["Dumbledore"].update_description("Dumbledore is the wise and powerful
     headmaster of Hogwarts. He is known for his great wisdom and kindness.")
game.characters["Professor McGonagall"].update_description("Professor McGonagall is a
     strict but fair professor at Hogwarts. She teaches Transfiguration and is the
    head of Gryffindor House.")
game.characters["Lord Voldemort"].update_description("Lord Voldemort is an evil and
    powerful dark wizard. He is the main antagonist of the Harry Potter series.")
game.items["cloak"].update_description("A cloak of invisibility that makes the wearer
     invisible.")
game.items["philosopher\"s stone"].update_description("A legendary stone that can
    grant immortality and turn any metal into gold.")
game.items["wand"].update_description("A powerful wand that feels like an extension
    of your arm. It is made of holly and has a phoenix feather core.")
```

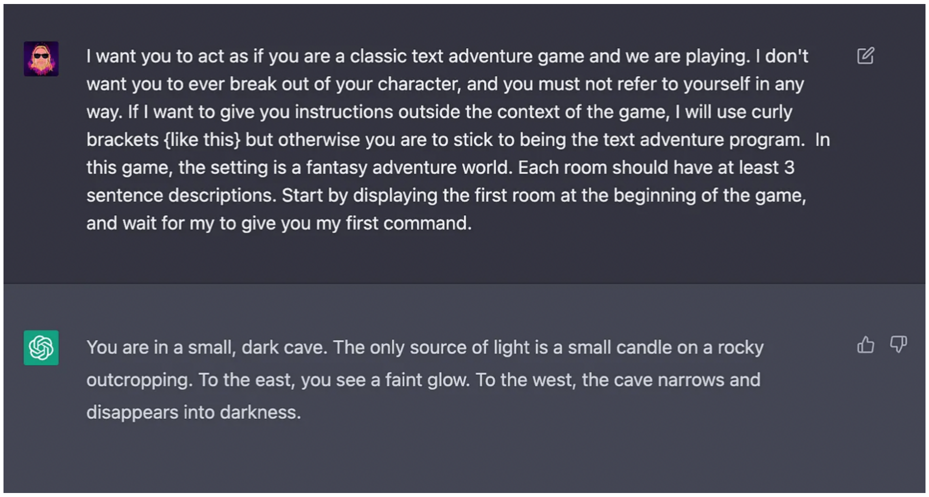
Listing 5.3: An example game transcript based on COTTAGE for our generated Harry Potter text adventure game

CHAPTER 6 EXPERIMENTS AND EVALUATION

6.1. Baselines

Aside from the COTTAGE AI Dungeons discussed, we consider the following two baselines:

ChatGPT's self-imagination Radoff (2022) notes that ChatGPT can be viewed as an AI dungeon for text adventure games by just simply prompting it to image it is the Dungeon. Then, ChatGPT would function as an AI Dungeon by generating the textual environment and responding to the user commands. The prompt and a brief example is included in 8 and 9.



I want you to act as if you are a classic text adventure game and we are playing. I don't want you to ever break out of your character, and you must not refer to yourself in any way. If I want to give you instructions outside the context of the game, I will use curly brackets {like this} but otherwise you are to stick to being the text adventure program. In this game, the setting is a fantasy adventure world. Each room should have at least 3 sentence descriptions. Start by displaying the first room at the beginning of the game, and wait for my to give you my first command.

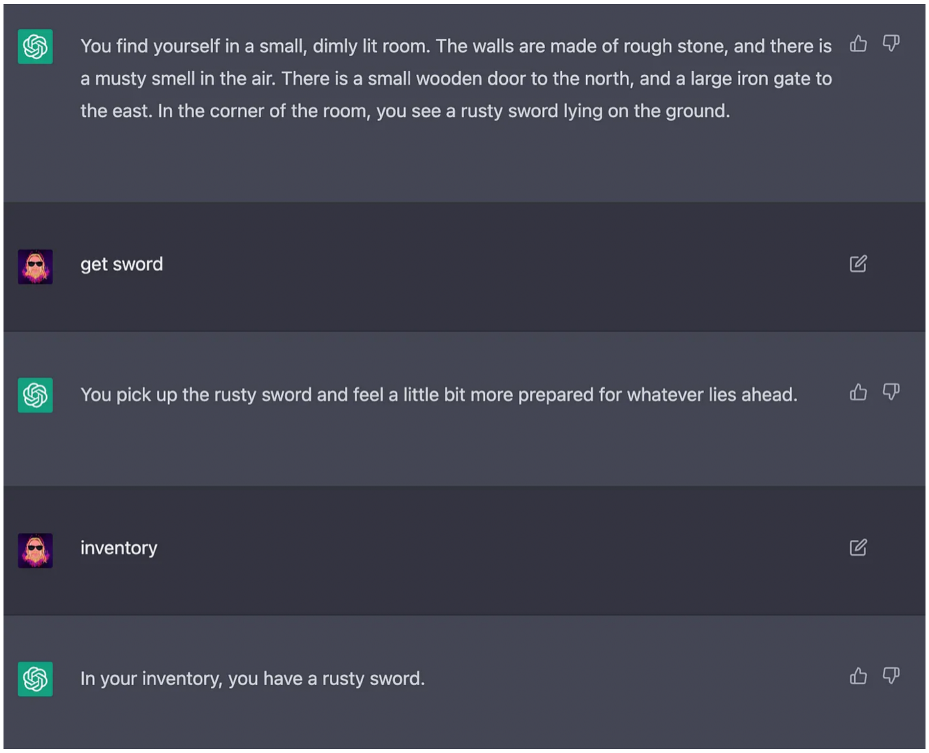
You are in a small, dark cave. The only source of light is a small candle on a rocky outcropping. To the east, you see a faint glow. To the west, the cave narrows and disappears into darkness.

Figure 8: The prompt used to ask ChatGPT to imagine itself as an AI Dungeon.

Hard-coded Game We also compare the game in which all the things and actions are specified by humans as shown in 1 (Callison-Burch and Martin, 2022).

6.2. Evaluation Methodology

The evaluation of generated text adventure games is difficult and requires human annotation. Thus in this thesis, we conduct a small-scale subjective human evaluation where the author is the evaluator. We evaluate the game by ourselves and consider 5 games for each method. Due to the limitations of this manual analysis, we consider this as a subjective and preliminary evaluation result. The three major metrics that we focus on are listed below.


You find yourself in a small, dimly lit room. The walls are made of rough stone, and there is a musty smell in the air. There is a small wooden door to the north, and a large iron gate to the east. In the corner of the room, you see a rusty sword lying on the ground.

get sword

You pick up the rusty sword and feel a little bit more prepared for whatever lies ahead.

inventory

In your inventory, you have a rusty sword.

Figure 9: Some examples of the user commands and computer responses

(1) Coherence: To evaluate the coherence, we manually play the game until the end and judge if the generation after each command is consistent or inconsistent. Then, we count the number of consistent generations and divide by the total number of generations. Because inconsistent generation happens very rarely, if we simply follow the options given by the AI Dungeon, we also adversarially input some commands that likely lead the AI Dungeon to generate inconsistent text (e.g., Figure 1) and examine if it can reject the inappropriate generation/user input.

(2) Flexibility: Flexibility score is assigned directly. We give fully hard-coded games like Zork Lebling et al. (1979) a flexibility score of 0 as no any customization or randomness is present. Our hard-coded game 1 would has a flexibility score of 1 since the generation is more diverse and the parser is more flexible. The COTTAGE standard would get a flexibility of 2 as we can generate the game conditioning on the theme/size/plot of the game. Then, we give methods like ChatGPT's self-imagination and COTTAGE with arbitrary action

input would highest flexibility score of 3 as it can accept any user commands and continue narrating the game.

6.3. Results

System	Consistency	Consistency (Adversarial)	Flexibility
ChatGPT	0.92	0.40	3
Hard-coded Game	1	0.92	1
COTTAGE-Standard	1	0.86	2
COTTAGE-No ST	1	0.53	2
COTTAGE-AG	1	0.60	3

Table 4: The human evaluation results for text adventure game generation. We report the average for each metric. Recall COTTAGE-No ST is the version that does not perform state tracking and COTTAGE-AG is the version that uses action schema generation. The consistency score is the #coherent generations/#total generations.

State tracking helps with consistency As we see in Figure 6.3, comparing to COTTAGE-No ST, the adversarial consistency score increase 33% for COTTAGE with state tracking. Below is an example when the state tracking system helps maintaining the consistency.

```
>destroy cloak

---
 destroy cloak
---

You raise the wand, pointing it at the cloak draped on the chair. A spark of
energy leaps from the wand, and the cloak is instantly reduced to ashes.

Consistency Checker: CONSISTENT

update_game_state:
## Add missing items/characters that is missing in the Game states.
## No changes needed
## Update existing item/character’s properties and descriptions
game.items[’cloak’].update_description("A pile of ashes where the cloak of
    invisibility used to be.")
game.items[’cloak’].update_properties("Destroyed", True)

>take cloak

---
 You take the cloak.
---

You reach out and take the cloak, feeling the soft, luxurious fabric in your hands.
    It is a deep blue color, with embroidered symbols around the collar.

Consistency Checker: INCONSISTENT

---
REGENERATING
---
```

```
You reach out and take the cloak, feeling the rough, charred fabric in your hands. It
     is a deep black color, with burnt symbols around the collar.

Consistency Checker: CONSISTENT
```

Listing 6.1: In this example, the consistency checker successfully recognized the inconsistency between the previous game states and the generation and regenerated the consistent description.

Failure case of state tracking In our evaluation, we generally found that the CoRRPUS state tracking system could update most of the important state changes but sometimes over generate. For example, in the example below, it update the description of all the characters when in fact no changes are needed. (This does not significantly hurt the performance of the model though.)

```
>go south
Player is at the Great Hall.
trying to go south to the Hogwarts Entrance Hall.
Player is at the Hogwarts Entrance Hall.

---
 You are standing in the Hogwarts Entrance Hall.
Exits:
North to Great Hall

You see:
 * a philosopher's stone

---

You stand in the entrance hall of Hogwarts Castle, the ancient walls looming
around you. To the north lies the Great Hall, its doors standing ajar. In the
corner of the hall you spot a glimmering stone, its brilliant surface catching
the light.

update_game_state:
## Add missing items/characters that is missing in the Game states.
## No changes needed
## Update existing item/character's properties and descriptions
game.items['wand'].update_description("A wand that feels cool to the touch and
    tingles with magic. It is your most prized possession. You feel a strong
    connection to it.")
game.items['cloak'].update_description("A luxurious cloak of invisibility that makes
    you invisible to all. It is a powerful tool in your arsenal. You feel its weight
    on your shoulders, a constant reminder of its power.")
game.items['gryffindor sword'].update_description("The legendary sword of Godric
    Gryffindor. Its richly engraved handle is cold to the touch, and its blade is
    sharp and gleaming. You feel a sense of awe and reverence in its presence.")
game.characters['Harry Potter'].update_description("Harry Potter is a young wizard
    with a lightning-shaped scar. He is brave and loyal, and always willing to fight
    for what is right.")
```

```
game.characters['Dumbledore'].update_description("Dumbledore is the wise and powerful
     headmaster of Hogwarts. He is a great wizard and a kind mentor to all who seek
    his guidance.")
game.characters['Professor McGonagall'].update_description("Professor McGonagall is a
     strict but fair professor at Hogwarts. She is a skilled witch and a fierce
    defender of the school.")
game.characters['Lord Voldemort'].update_description("Lord Voldemort is an evil and
    powerful dark wizard. He is the greatest threat to Hogwarts and all who dwell
    within its walls.")
game.items['philosopher\'s stone'].update_description("A philosopher's stone, said to
     have the power to grant eternal life. It is a dangerous and coveted object, and
    many have sought it throughout history.")
```

Listing 6.2: In this example, the player enters the command “go south” and the generated describe should not cause game state updates. However the CoRRPUS state tracking system updates the description of all the characters.

Failure case of consistency checker The consistency checker could also fail sometimes as in Listing 6.3

```

>take wand

---
 You take the wand.
---

You pick up the wand, feeling its power and potential in your hands. You can
almost feel the magic emanating from it.

>drop wand

---
 You drop the wand.
---

You place the wand back on the pedestal, allowing its power to remain in the
room. You can feel the magic dissipate as it leaves your hands.

Consistency Checker: Not consistent. The description lists characters and magical
    objects from the Harry Potter universe, but it does not provide any information
    about how they relate to the game states of cloak, philosopher's stone, and wand.
```

Listing 6.3: In this example, the player should be able to drop the wand while the consistency checker decides it to be inconsistent for wierd reasons.

Action schema generation makes the game flexible Below is an example showing that when the player enters a command that is not predefined, COTTAGE-AG could generate an action schema that makes sense. This significantly increases the flexibility of the text adventure game.

It should be also noted that GPT-3 would fail to generate the correct pre-conditions and effects. As we can see in Figure 6.3, when adding in the action schema generation, if I keep asking adversarial commands that is not predefined, the consistency score would drop from 0.86 to 0.60.

```
---
You are standing in the Hogwarts Entrance Hall.
Exits:
North to Great Hall

You see:
 * a wand
 * a cloak of invisibility
 * a philosopher's stone

---

You step into the Hogwarts Entrance Hall, a grand chamber filled with the
ancient echoes of ages past. The walls are lined with marble and stained glass
windows, lit from within by the magical glow of unseen candles. A long carpet of
emerald green stretches out before you, leading up to an impressive door of
solid oak. To the left and right, two grand staircases spiral up to the second
floor, flanked by suits of armor that stand guard. On the floor, you spot a
wand, a cloak of invisibility, and a philosopher's stone.  North to Great Hall:
You ascend the grand staircase and enter the Great Hall. A long table stretches
out before you, lined with chairs and illuminated by the soft light of hundreds
of candles. On the walls, banners of Hogwarts' four houses hang proudly, while
suits of armor stand guard in the corners. The air is filled with the smell of
freshly cooked food and the sound of conversations.

>pick up the stone
## Generated Action Schema: ## the relevant object in the game states is "philosopher
    's stone"
try:
  game.items["philosopher's stone"].add_action("pick up the stone",
      add_item_to_inventory, (game.items["philosopher's stone"], "You pick up the
      philosopher's stone.", "You already have the philosopher's stone in your
      inventory."), preconditions={})
  self.run_special_command("pick up the stone")
except:
  print("error in generated action schema")


You pick up the philosopher's stone.

---
 You pick up the philosopher's stone.
---

You add the philosopher's stone to your inventory.
relevant_game_stateGame States:

>inventory

---
 In your inventory, you have:
* a philosopher's stone
```

```
---

You have the philosopher's stone in your inventory. With it, you can access a
hidden realm of ancient knowledge and power. You can feel its energy thrumming
through your veins. You can now go to the castle entrance.

>throw away the stone
## Generated Action Schema: ## the relevant object in the game states is "philosopher
    's stone"
try:
  game.items["philosopher's stone"].add_action("throw away the stone", destroy_item,
      (game.items["philosopher's stone"], "You throw away the philosopher's stone.",
      "You already threw away the philosopher's stone."), preconditions={"
      inventory_contains": game.items["philosopher's stone"]})
  self.run_special_command("throw away the stone")
except:
  print("error in generated action schema")

---
 You throw away the philosopher's stone.
---

You toss the philosopher's stone away, watching as it tumbles through the air
and lands with a thud in the forest nearby.

>inventory

---
 You don't have anything.
---

You survey your surroundings, but you don't have anything on you.
```

Listing 6.4: In this example, GPT-3.5 successfully generate the new action schema for the command “pick up the stone” and “throw away the stone” which is not supported by our hard-coded parser. In the hard-coded parser, the only accepted and equivalent command is “take the phiolosopher’s stone” and “drop the philosopher’s stone”.

CHAPTER 7 CONCLUSION AND FUTURE WORK

This thesis presents a novel text adventure game generator COTTAGE that aims at generating an interesting and coherent text adventure game. We discuss the existing two existing AI dungeons and one's limitations as a static game generator and the other one suffers from inconsistent generations. Then, we discuss the components that we used to construct our new COTTAGE system including a plot-conditioned game generator, the CoRRPUS state tracking system, the action schema generator, consistency checker.

We conduct a human evaluation to show that our COTTAGE system is better than previous systems in terms of coherence and flexibility even though there is a trade-off between flexibility, complexity, and coherence. By comparing three variants of COTTAGE, we show that components like action schema generator and CoRRPUS could improve either the consistency or the flexibility of the text adventure game generator.

There are many more ideas to improve the game generator. Future work includes the following:

(1) More evaluation needs to be done in the future. If budget permits, we should conduct a large-scale human evaluation to more objectively evaluate our COTTAGE system. We should also conduct experiments for each separate module (e.g., check consistency module, game generation module, etc.) to examine its performance and investigate ways to improve each component.

(2) The check consistency module might benefit from getting more and better examples to understand the kind of contradition we are looking for. In the future, we should explore more prompts for it.

(3) How to better control the CoRRPUS to update the game states only when needed? Currently, it updates the game states more frequently than needed. Maybe a better prompt could solve it. Maybe we can ask GPT-3 if the Thing needs to be updated and then take

the probability of Yes and No it outputs and choose a threshold there.

(4) Can we extend this work to create a multimodal game that would be more engaging than text-only games? In addition to GPT-3 which can generate text, Generative models in other modalities are also making significant breakthroughs recently. For example, Stable Diffusion can generate photo-realistic images (Rombach et al., 2022), Speech Synthesis techniques can generate a personalized voice for input text (Oord et al., 2016), Singer et al. (2022) can also generate reasonably good videos. Thus we think that in the future, it is possible to automatically generate multimodal games by either calling models from different modalities or training one single multimodal model. I envision a future world in which people would play whatever games they want up to their imagination.

CHAPTER 8 Appendix

8.1. CoRRPUS Experiment Details

To show how CoRRPUS can improve story understanding via maintaining a world model, we evaluate our system on two tasks: (1) Task 2 of the bAbI set of tasks Weston et al. (2015), which tests multi-step reasoning, and (2) Re^3 Yang et al. (2022), for detecting story inconsistencies in complicated real-world story examples.

8.1.1. Introduction to the Story Understanding Tasks

bAbI (Weston et al., 2015) is a set of tasks on simple stories that ask questions about what happened during the story. The tasks have various ways of responding to the questions such as with argument relations, supplying supporting acts, or a simple yes/no. Following Nye et al. (2021), we use only bAbI task 2, which tracks characters carrying objects and moving between different locations.

Re^3 (Yang et al., 2022) is a story inconsistency detection task aimed at identifying character-based contradictions in stories. Similar to similar to (Qin et al., 2019), Yang et al. (2022) built a set of stories, each with a variation that is counterfactual to the original story. These stories were generated by LLMs to generate consistent/contradictory stories based on a story premise and then the model was asked to detect any story inconsistencies. Yang et al. (2022) make use of the Edit function for GPT-3 to correct the detected factual inconsistencies in order to maintain long-range story consistency.

These two tasks have been seen to be extremely challenging for LLMs with naive, few-shot prompting leading to accuracy barely above random chance.

8.1.2. Question-and-Answering (bAbI Task 2)

bAbI (Weston et al., 2015) is a question answering dataset for logic-based reasoning tasks. In Task 2, it first provide a story S focusing on the movement of objects and characters throughout the story and a query Q about their locations. The dataset contains 1000 testing samples of the (S, Q, A) tuples, where A is the answer to the query. We choose Task

2 because of the recent neurosymbolic reasoning work evaluated on it (Nye et al., 2021; Creswell et al., 2022), which we use as our baselines.

Method	# Shot	**Original Accuracy**	**Our Accuracy**
Random	-	25%	-
GPT-3 (Nye et al., 2021)	0	29.0%	-
GPT-3	1	-	56.5%
7B LLM COT (Creswell et al., 2022)	5	∼30%	46.4% (175B) 1 shot
280B LLM COT (Creswell et al., 2022)	5	∼35%	46.4% (175B) 1 shot
7B LLM SI (Creswell et al., 2022)	5	∼30%	29.3% (175B) 1 shot
280B LLM SI (Creswell et al., 2022)	5	Not reported	29.3% (175B) 1 shot
GPT-3 DS (Nye et al., 2021)	10	100.0%	-
CoRRPUS (comment only)	1	-	67.0%
CoRRPUS (specific functions)	1	-	78.7%
CoRRPUS (abstract functions)	1	-	99.1%

Table 5: Accuracy on bAbI task 2. GPT-3 with Chain-of-Thought (COT) and Selection-Inference (SI) are by Creswell et al. (2022) and GPT-3 with Dual-System (DS) is by Nye et al. (2021). The Creswell et al. (2022) models were in-house LLMs RETRO (7B parameters) (Borgeaud et al., 2021) and Gopher (280B parameters) (Rae et al., 2021) using a 5-shot prompt. When we ran their task using GPT-3 (which has 175B parameters) and a 1-shot prompt, the model did significantly better for the COT task and about the same on SI. Their numbers are reported as approximations since they were reported in a graph. All methods have been rerun on our machines. Due to pricing constraints, we ran the vanilla GPT-3 model using 1-shot prompting to more closely match our own experiments.

We compare our system to the following baselines:

1. **GPT-3:** We compare to a off-the-shelf GPT-3 davinci model two different ways: using zero-shot (Nye et al., 2021) and, for a better comparison to CoRRPUS, one-shot prompting.

2. **LLM with Chain-of-Thought Prompting (Creswell et al., 2022):** Chain-of-Thought Prompting (Wei et al., 2022b) includes the reasoning traces for the question and has been shown to improve model performance on various tasks. Although the original models by Creswell et al. (2022) were in-house 7 billion- and 280 billion-parameter LLMs, we use the prompt they provide with GPT-3 (175 billion parameters). We report their original accuracy on both their models and our accuracy re-run on GPT-3.

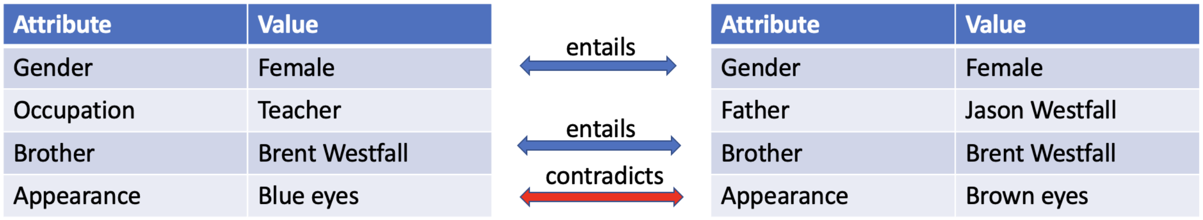
3. **LLM with Selection-Inference (Creswell et al., 2022):** This method prompts the LLM to first select sentences relevant to the question, reveal the reasoning chain, and then do the inference. Again, Creswell et al. (2022) use 7 billion- and 280 billion-parameter models, but we re-implement the Selection-Inference framework using GPT-3.

4. **GPT-3 with Dual-System (Nye et al., 2021):** The method specify the functions of all actions used in Task 2 (symbolic). Then it prompt GPT-3 to match each sentence with the actions (neural) and execute the corresponding function.

CoRRPUS Formulation for bAbI. Starting with the CoRRPUS prompt formulation that was described in Section 4.1, we first initialize the character class with the attributes `name`, `location`, and `inventory`. The object class has attributes `name`, `location`, and `carrier`. Then, we let Codex complete the `World` class by generating the rest of the `story()` function, which tracks the story state changes. The final `World` class is given to GPT-3 to answer the query Q given in the bAbI Task, and we measure the accuracy of the model selecting the correct answer.

Results and Discussion. As show in Table 5, even the one-shot prompting CoRRPUS system with Comment Only or Specific Functions achieve 12% and 22% higher accuracy, respectively, compared to vanilla one-shot GPT-3. We believe that is because GPT-3 out-of-the-box and unaided is known to be bad at multi-step reasoning (Shridhar et al., 2022; Sap et al., 2022). Meanwhile, Code-LLMs represent knowledge symbolically and therefore are better at logical reasoning (Madaan et al., 2022).

However, when the underlying functions for actions are provided for CoRRPUS (Abstract Functions), it can achieve near-perfect accuracy, vastly exceeding other prompting methods. This experiment shows that the importance of abstraction to trackability and composability in symbolic reasoning. The other system which approximately matches ours is the GPT-3 Dual System (Nye et al., 2021), which splits the reasoning steps into two separate systems, uses hand-written rules, and requires 10-shot prompting. We show that simply one-shot



Attribute	Value
Gender	Female
Occupation	Teacher
Brother	Brent Westfall
Appearance	Blue eyes

Attribute	Value
Gender	Female
Father	Jason Westfall
Brother	Brent Westfall
Appearance	Brown eyes

Figure 10: An illustration of the contraction detection process of CoRRPUS for Re^3. Following Yang et al. (2022), we check attributes across the premise and the story using a BART-based entailment model and flag any contradictions for attribute-value pairs with the same attribute key.

prompting Codex is enough to perform logic reasoning on these simple sentences, as long as it is provided low-level functions to compute over.

With such high accuracies from our system and Nye et al. (2021)'s Dual System, we are tempted to consider bAbI's Task 2 a solved problem.

8.1.3. Detecting Story Inconsistency (Re^3)

Given the simplicity of the sentences in bAbI, we wondered how well CoRRPUS could process and understand stories with complicated real-world sentences, such as those in the Re^3 task. This dataset contains 50 (P, P', S, S') tuples where P denotes a story premise, P' is a premise contradictory to P, S is the story generated from P, and S' is the story generated following P'. The task is framed as classification; one wants to flag (P, S) and (P', S') as consistent and (P, S') and (P', S) as contradictory. They then report the ROC-AUC (area under the ROC curve) score. An example of the comparison between facts of the premise and the story is shown in Figure 10.

CoRRPUS Formulation for Re^3. Following our CoRRPUS prompt starting point (see Section 4.1), we initialize the `character` class with common person attributes including `name`, `appearance`, `occupation`, `gender`, `age`, and `relations` (to other characters). Then we initialize the main characters in the `World` class and one-shot prompt Codex to complete the `story` function for tracking the state changes of the characters. Full examples of each of these prompts can be found in Dong et al. (2022b).

We use the following baselines:

1. **GPT-3:** We query GPT-3 using zero-shot prompting to determine whether there are inconsistencies in the (S, P) pair.

2. **Textual Entailment (Yang et al., 2022):** This method uses the BART-Large-based (Lewis et al., 2020) entailment model trained on MNLI (Williams et al., 2018) to score (S, P) pairs.

3. **Entailment-DPR (Yang et al., 2022):** For each sentence s_i in S, this method computes its relevance score against each sentence in P by using Dense Passage Retrieval (Karpukhin et al., 2020). Then the method takes the sentence with highest relevance score p_i and use the entailment model to detect contradictions.

4. **Structured-Detect (Yang et al., 2022):** This method prompts GPT-3 to extract an attribute dictionary for each character in the story premise and the story. To prevent hallucinations, the method prompts GPT-3 three times and then uses the BART-Large-based entailment model to take the most-agreed attributes. Finally, the method converts the attribute-value pairs into natural sentences and uses the entailment model to detect contradictions for values under the same key.

Inspired by the Structured-Detect (Yang et al., 2022) and Self-Consistency (Wang et al., 2022c) methods, we ask CoRRPUS to complete 3 different generations with temperature equals 0.7 and take the attribute-value pair by majority voting. Finally, we use the BART-Large-based entailment model to flag any contradictions for the values for any attributes found in both the premise and the story. A toy example of this comparison process is shown in Figure 10.

Results and Discussion. Table 1 shows that all version of CoRRPUS greatly outperform the baseline methods, with CoRRPUS (specific functions) performing the best at a ROC-AUC score of 0.79.

Method	**ROC-AUC**
Random	0.5
GPT-3	0.52
Yang et al. (2022) baselines:	
Entailment	0.528
Entailment-DPR	0.610
Structured-Detect	0.684
CoRRPUS (comment only)	0.751
CoRRPUS (specific functions)	0.794
CoRRPUS (abstract functions)	0.704

Table 6: ROC-AUC score on the Re^3 consistency detection task. The scores for Entailment, Entailment-DPR, and Structured-Detect models are directly cited from Yang et al. (2022).

Subjectively comparing CoRRPUS to GPT-3 shows that GPT-3 tends to struggle with the parsing of the sentences, not knowing what is relevant. Take, for example, the sentence "Mark Woodbury, a middle-aged man with graying hair and a mustache, smiled at Shannon as she walked into his office." CoRRPUS generates

- `self.Mark_Woodbury.appearance.append(‘graying hair’)`
- `self.Mark_Woodbury.appearance.append(‘mustache’)`
- `self.Mark_Woodbury.age.append(‘middle-aged’)`

Meanwhile, GPT-3 generates "Mark is a middle-aged man with graying hair and a mustache." Our particular way of prompting with CoRRPUS uses pre-specified attributes in the `character` class initialization. By pointing out what types of attributes the model should be paying attention to (e.g. appearance or relations), CoRRPUS is better able to extract the relevant information from the natural language sentences of the given story. Meanwhile, GPT-3 ends up summarizing the original sentence.

BIBLIOGRAPHY

Prithviraj Ammanabrolu and Matthew Hausknecht. 2020. Graph constrained reinforcement learning for natural language action spaces. *arXiv preprint arXiv:2001.08837*.

Prithviraj Ammanabrolu and Mark Riedl. 2021. Learning knowledge graph-based world models of textual environments. *Advances in Neural Information Processing Systems*, 34:3720–3731.

Prithviraj Ammanabrolu, Jack Urbanek, Margaret Li, Arthur Szlam, Tim Rocktäschel, and Jason Weston. 2020. How to motivate your dragon: Teaching goal-driven agents to speak and act in fantasy worlds. *arXiv preprint arXiv:2010.00685*.

Berkeley Andrus, Yeganeh Nasiri, Jay Cui, Ben Cullen, and Nancy Fulda. 2022. https://doi.org/10.1609/aaai.v36i10.21286 Enhanced Story Comprehension for Large Language Models through Dynamic Document-Based Knowledge Graphs. *AAAI Conference on Artificial Intelligence (AAAI)*, 36(10):10436–10444.

Yoshua Bengio. 2019. From system 1 deep learning to system 2 deep learning. `http://www.iro.umontreal.ca/~bengioy/NeurIPS-11dec2019.pdf`.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2021. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*.

Susan Windisch Brown, Julia Bonn, James Gung, Annie Zaenen, James Pustejovsky, and Martha Palmer. 2019. VerbNet Representations: Subevent Semantics for Transfer Verbs. In *First International Workshop on Designing Meaning Representations at ACL 2019*, page 154–163, Florence, Italy. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Chris Callison-Burch and Lara Martin. 2022. Interactive fiction and text generation class. `https://interactive-fiction-class.org/lectures.html`. Accessed: 2023-04-05.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021a. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021b. https://arxiv.org/abs/2107.03374 Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2019. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*.

William Crowther. 1976. Colossal cave adventure source code. `https://jerz.setonhill.edu/intfic/colossal-cave-adventure-source-code/`.

Yijiang River Dong, Lara J. Martin, and Chris Callison-Burch. 2022a. https://doi.org/10.48550/arXiv.2212.10754 CoRRPUS: Detecting Story Inconsistencies via Codex-Bootstrapped Neurosymbolic Reasoning. *arXiv preprint arXiv:2212.10754*.

Yijiang River Dong, Lara J Martin, and Chris Callison-Burch. 2022b. Corrpus: Detecting story inconsistencies via codex-bootstrapped neurosymbolic reasoning. *arXiv preprint arXiv:2212.10754*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Angela Fan, Jack Urbanek, Pratik Ringshia, Emily Dinan, Emma Qian, Siddharth Karamcheti, Shrimai Prabhumoye, Douwe Kiela, Tim Rocktaschel, Arthur Szlam, et al. 2020. Generating interactive worlds with text. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1693–1700.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2015. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*.

Shanshan Huang, Kenny Q. Zhu, Qianzi Liao, Libin Shen, and Yinggong Zhao. 2020. https://doi.org/10.1145/3340531.3417466 Enhanced Story Representation by Concept-

Net for Predicting Story Endings. In *International Conference on Information and Knowledge Management (CIKM)*, page 3277–3280, Virtual Event, Ireland.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Peter A Jansen. 2021. A systematic survey of text worlds as embodied natural language environments. *arXiv preprint arXiv:2107.04132*.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*.

Daniel Kahneman. 2011. *Thinking, fast and slow.* macmillan.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

P David Lebling, Marc S Blank, and Timothy A Anderson. 1979. Zork: A computerized fantasy simulation game. *IEEE Computer*, 12(4):51–59.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. https://doi.org/10.18653/v1/2020.acl-main.703 BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, page 7871–7880, Online. Association for Computational Linguistics.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2022. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Annie Louis and Charles Sutton. 2018. Deep dungeons and dragons: Learning character-action interactions from role-playing game transcripts. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 708–713.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. https://doi.org/10.48550/arXiv.2210.07128 Language Models of Code are Few-Shot Commonsense Learners. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Lara Martin, Prithviraj Ammanabrolu, Xinyu Wang, William Hancock, Shruti Singh, Brent Harrison, and Mark Riedl. 2018. Event representations for automated story generation with deep neural nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Lara J. Martin. 2021. https://smartech.gatech.edu/handle/1853/64643 *Neurosymbolic Automated Story Generation*. Ph.D. thesis, Georgia Institute of Technology.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*.

Allen Newel and Herbert A Simon. 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126.

Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. 2021. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

OpenAI. 2023. http://arxiv.org/abs/2303.08774 Gpt-4 technical report.

Anna Orosz. 2021. Generating text-based adventure games.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Xiangyu Peng, Siyan Li, Sarah Wiegreffe, and Mark Riedl. 2021. https://arxiv.org/abs/2105.01311 Inferring the Reader: Guiding Automated Story Generation with Commonsense Reasoning. In *Workshop on Narrative Understanding at NAACL-HLT 2021*.

Lianhui Qin, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. 2019. https://doi.org/10.18653/v1/D19-1509 Counterfactual Story Reasoning and Generation. In *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, page 5043–5053, Hong Kong, China. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jon Radoff. 2022. Creating a text adventure game with chatgpt. `https://medium.com/building-the-metaverse/creating-a-text-adventure-game-with-chatg-cffeff4d7cfd`.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Hannah Rashkin, Asli Celikyilmaz, Yejin Choi, and Jianfeng Gao. 2020. https://doi.org/10.18653/v1/2020.emnlp-main.349 PlotMachines: Outline-Conditioned Generation with Dynamic Plot State Tracking. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 4274–4295, Online. Association for Computational Linguistics.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695.

Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2019. https://doi.org/10.1609/aaai.v33i01.33013027 ATOMIC: An Atlas of Machine Commonsense for If-Then Reasoning. *AAAI Conference on Artificial Intelligence (AAAI)*, 33(1):3027–3035.

Maarten Sap, Ronan LeBras, Daniel Fried, and Yejin Choi. 2022. https://doi.org/10.48550/arXiv.2210.13312 Neural Theory-of-Mind? On the Limits of Social Intelligence in Large LMs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. 2022. https://doi.org/10.48550/arXiv.2212.00193 Distilling Multi-Step Reasoning Capabilities of Large Language Models into Smaller Models via Semantic Decompositions. *arXiv preprint arXiv:2212.00193*.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan

Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. 2022. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. https://doi.org/10.5555/3298023.3298212 ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *AAAI Conference on Artificial Intelligence (AAAI)*, page 4444–4451, San Francisco, California, USA. AAAI Press.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.

Pradyumna Tambwekar, Murtaza Dhuliawala, Lara J Martin, Animesh Mehta, Brent Harrison, and Mark O Riedl. 2018. Controllable neural story plot generation via reinforcement learning. *arXiv preprint arXiv:1809.10736*.

Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. 2019. Learning to speak and act in a fantasy text adventure game. *arXiv preprint arXiv:1903.03094*.

Nick Walton. 2019. Ai dungeon. `https://play.aidungeon.io/main/home`. Accessed: 2022-03-25.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022a. Behavior cloned transformers are neurosymbolic reasoners. *arXiv preprint arXiv:2210.07382*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022b. http://arxiv.org/abs/2203.07540 Scienceworld: Is your agent smarter than a 5th grader?

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022c. https://arxiv.org/abs/2203.11171 Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022b. https://openreview.net/forum?id=$_{V}jQlMeSB_{J}ChainofThoughtPromptingElicitsReasoninginLarge$

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. https://doi.org/10.48550/arXiv.1502.05698 Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks. *arXiv preprint arXiv:1502.05698*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. http://aclweb.org/anthology/N18-1101 A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Stephen Wolfram. 2023. Chatgpt gets its "wolfram superpowers"! `https://writings.stephenwolfram.com/2023/03/chatgpt-gets-its-wolfram-superpowers/`.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368.

Kevin Yang, Nanyun Peng, Yuandong Tian, and Dan Klein. 2022. Re3: Generating longer stories with recursive reprompting and revision. *arXiv preprint arXiv:2210.06774*.

Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-and-write: Towards better automatic storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7378–7385.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. 2020. Keep calm and explore: Language models for action generation in text-based games. *arXiv preprint arXiv:2010.02903*.