Language-Driven Play: Large Language Models as Game-Playing Agents in Slay the Spire

Bahar Bateni
bbateni@ucsc.edu
University of California Santa Cruz
Santa Cruz, USA

Jim Whitehead
sjw@ucsc.edu
University of California Santa Cruz
Santa Cruz, USA

ABSTRACT

One of the major challenges in procedural generation of game rules is evaluating the generated content. Since the effect of a rule on game balance and complexity might not be immediately apparent, one way to evaluate such a content is to simulate the gameplay. To achieve this, it is necessary to create an agent capable of both playing the game and adjusting to alterations in the game's design, which is often referred to as a general game-playing agent.

In this paper, we study the ability of Large Language Models (LLMs) in performing this task. Our focus centers on a simplified implementation of the card game Slay the Spire, in which the rules support a wide variety of interesting and complex interactions between the cards. We analyze the performance of LLMs in understanding the cards and their synergies based solely on their description. The performance of the LLM agent is evaluated against alternative game-playing agents across diverse scenarios. Our findings reveal that although the LLM agent may not be optimized for finding the best move, it exhibits superior long-term planning without necessitating specialized training.

KEYWORDS

Procedural Content Generation, Rule Generation, General Game-playing, Large Language Models

ACM Reference Format:
Bahar Bateni and Jim Whitehead. 2024. Language-Driven Play: Large Language Models as Game-Playing Agents in Slay the Spire. In *Proceedings of the 19th International Conference on the Foundations of Digital Games (FDG 2024), May 21–24, 2024, Worcester, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3649921.3650013

1 INTRODUCTION

Procedural Content Generation (PCG) systems serve as invaluable tools for creating diverse game content such as images, textures, audio, levels, etc. They can also be used to shape a game's design by generating or modifying the ruleset [1, 5, 12, 17]. However, this can prove challenging, as tweaking these rules can significantly impact gameplay dynamics, balance, and overall difficulty, often in ways not immediately understood in the generation process.

In a taxonomy of search-based PCG, Togelius et al. describe three possibilities for evaluating the content generated by a search-based PCG system [18]. First, a direct evaluation of the content involves defining domain specific metrics which can be directly evaluated without playing the game (e.g. the number of paths to exit a maze). Second, simulation-based evaluation is based on an artificial agent playing through the game. Finally, interactive evaluation is calculated during gameplay and based on interactions with a human player. Of these three, simulation-based evaluation is well suited when the generation process involves modifying the game rules. The existence of a human in the loop of interactive evaluation makes them unfit for integration in a PCG system, at least when the content is not near its final state. On the other hand, although direct evaluation methods are less computationally expensive than simulation-based methods, their domain specific nature often renders them less suited for assessing content with unpredictable and complex effects such as game rules.

Crucially, this process requires a game playing agent with the ability to respond to alterations in the rules. In other words, the agent should receive the set of rules as part of its input and still be able to play the game reasonably well. Such an agent is often referred to as a general game playing agent (GGP), as described by Pell [10].

Furthermore, we are interested in general game playing for card games. We choose the card game *Slay the Spire* [3], or more accurately a simplified implementation of it which we call *MiniStS*, as our experimental environment. The turn-based and single-player characteristics of the game, as well as the vast space of possibilities for designing new cards and their interactions make this a flexible and expressive test environment for our case. We further discuss these characteristics in section 3. Moreover, general game playing in card games is a less-explored and interesting challenge by itself. Since the cards are defined by natural language, they provide a way to add arbitrary rule sets to the game by introducing new cards. While generating the cards can be done by using NLP techniques[7, 14], evaluating them requires a careful consideration of game balance. This delicate balance can be thrown off with any new card not only because of the properties of that card by itself, but also because of its synergies with other elements in the game.

Simultaneously, the development of large language models (LLMs) has revolutionized the landscape of artificial intelligence research. The concept of employing a generalized pre-trained model, trained on large datasets, has become a cornerstone for various applications. These models exhibit a remarkable capacity to tackle a diverse array of tasks with a minimal need for specialized training. Large language models, by design, possess an inherent adaptability that

FDG 2024, May 21–24, 2024, Worcester, MA, USA

ACM ISBN 979-8-4007-0955-5/24/05
https://doi.org/10.1145/3649921.3650013

enables them to traverse a broad range of domains, showcasing their versatility in handling challenges.

This paper delves into the intersection of these two domains, proposing a novel fusion that capitalizes on the strengths of large language models, especially their generalizability, in addressing the challenge of general game playing. Specifically, the aim is to leverage a pre-trained language model as an agent capable of receiving game rules as input and generating the appropriate moves as output. The inherent generality of the language models raises the exciting prospect of creating an agent that can seamlessly adapt to a wide range of rule sets while maintaining acceptable performance.

We showcase this ability in future sections by analyzing a set of experiments in the specific context of playing the card game Slay the Spire. We choose this context based on a set of desirable properties which we discuss in section 3. By conducting these experiments, our research aims to contribute to the development of adaptable and proficient agents in dynamic gaming environments with the goal of facilitating automated evaluation of game rules.

2 RELATED WORK

2.1 Automated Game Design

Simulation-based evaluation requires specialized agents capable of adapting to changes in the ruleset. An early exemplar of such an agent is METAGAMER [11]. Following Pell's description of METAGAME [10] and the introduction of the general game playing challenge, METAGAMER is an AI agent that can play the grammatically generated chess-like games in the METAGAME-specific language. While Pell initially envisioned METAGAME as an environment to assess the performance of METAGAMER in general game playing (and not to test METAGAME's game generation abilities through METAGAMER), it nevertheless stands as one of the earliest instances of an agent equipped to handle abstract game mechanics.

Later, Browne and Maire introduce Ludi, a search-based general game system capable of generating new games defined by the Ludi game description language and evaluating them through self-play [1]. The Ludi GGP, which the authors refer to as the core of Ludi, is able to play any generated game. This is made possible by first interpreting the game rules (e.g. listing all legal moves, checking for terminal states, etc.) and then using an alpha-beta adversarial search with iterative deepening to choose the best move. The estimated value of non-terminal states are provided by a strategy module, which uses a weighted combination of scores offered by 20 advisors focused on aspects such as mobility, attacking potential, etc. Our proposed backtrack agent, which we discuss further in section 4.2, is inspired by this method.

While the aforementioned characteristics of simulation-based evaluation methods make them appropriate for application in a PCG context, it's important to acknowledge alternative approaches to ruleset generation. One such system is exemplified by Smith and Mateas in Variations Forever [12]. By representing rulesets in logical terms, Variations Forever is able to use answer-set programming (ASP) to generate valid rulesets. The use of ASP here allows for restricting the design space to certain playability constraints (e.g. having a player character) and even zooming in on a subspace of interesting games (for example, by rejecting co-occurrence of mechanics known to interact poorly).

While we focus on agents which can play the game without any specialized training on the rules, one can imagine certain advantages in learning how to play with every ruleset from scratch. Togelius and Schmidhuber cleverly use this to measure the learnability of the game [17]. Evolving controller agents for each generated ruleset offers a metric of game complexity based on how quickly the game is learnt by the agent. Another advantage to training agents from scratch can be better performance for the agents. Mechanic Maker 2.0 [5] uses an RL agent to train for each set of generated rules. The generated rules are created in an iterative process, in which the RL agent is trained from scratch for every new set of rules. Then, based on properties of the agent such as the number of steps taken to reach the goal or the number of times a specific rule was used, a fitness value is created for the set of rules. A search-based method is used based on this fitness value to find appropriate rulesets in the design space.

In contrast to the referenced work in this section, a key distinction of our experiments lies in the nature of the design space and the means through which it is defined. Part of the power of these previously mentioned systems comes from the clever definition of their design space through a Game Description Language (GDL). In our specific domain of card games, however, the challenge is amplified as cards are described in open-ended terms using human natural language. Unlike systems relying on GDLs, incorporating LLMs enables our agent to comprehend and reason about the game rules directly from the natural language description of cards. Our approach not only allows our agent to perform general game playing in the context of card games but also enable the full extent of creative freedom in card design. This can be done by integrating this agent with systems designed to generate natural language description of cards such as RoboRosewater [7] and models with the ability to translate these descriptions into executable code [6].

2.2 Automated Card Generation

There is a small but growing literature on on card generation systems. Several systems in this literature explore the generation of cards in collectible card games, yet the majority of them lack an integrated evaluation process, with their primary emphasis lying on fostering creativity in card generation [7, 14]. An exemplar is RoboRosewater, an automated card generation system relying on a deep recurrent neural network [7]. RoboRosewater solely focuses on imitating the text representation of cards in Hearthstone without considerations for game balance or interactions between cards. This results in generation of interesting and creative but unbalanced or occasionally unplayable cards.

Mystical Tutor stands out as one of the few mixed-initiative card generation tools[14]. The system employs a Long Short-term Memory (LSTM) model to complete partial descriptions provided by the user, a task framed as a sequence-to-sequence (seq2seq) translation. Evaluation metrics are focused on the performance of the translation task, with an additional set of aesthetic metrics whose computation relies on human judgement.

Chaos Cards [2] provide a grammatical model for generating card description in the style of Hearthstone. Chen and Guy employ an RNN for predicting the strength of the cards to evaluate them. The network is trained during the play based on the actions of the general game playing agent. To balance effectiveness and performance, the general game playing agent uses a one turn look-ahead searched-based AI that evaluates the next state of the game for every possible action. This agent is enhanced by using the RNN network along its look-ahead process. Additionally, many considerations are taken into account to simulate the unknown nature of the opponent's hand as well as the game meta and the deck-building aspect of the game. As we discuss in the section 3, we evade these complexities by using a single player card game as our testbed. Furthermore, inspired by the general game playing agent used here, we use a 4 turn look-ahead agent which is used as part of our experiment for comparison with the LLM agent.

2.3 Large Language Models in PCG

The use of LLMs in the context of PCG has grown substantially in recent years. Todd et al. propose a level generation process based on using LLMs as the generative component [16]. Although the focus of our work is on using LLMs for evaluation purposes, and more specifically as game playing agents, we find their work relevant since it also relies on LLMs understanding game rules. Similarly, marioGPT uses a fine-tuned version of GPT2 to generate tile-based levels for Super Mario Bros [13].

Closer to our use case, LLMs have been used for playing different games. Noever et al. introduce chess transformer, a GPT2 model fine-tuned on playing chess [8]. Their findings show that the fine-tuned model can play plausible strategies and display an understanding of classic openings. One major difference with our work is that the agent here is specifically trained on playing chess, both in the pre-training and in the fine-tuning process. While this can result in playing chess with a higher level of expertise, it is not appropriate for our case since we are aiming to use the agent in an environment with ever-changing rules.

Voyager is another example of using LLMs for playing games, specifically Minecraft [19]. In this case, Voyager uses a growing skill-library that provides the agent with the ability to constantly add or retrieve complex skills while learning. Voyager uses GPT-4 by sending prompts containing 4 components: encouragement for diverse behaviour, the agent's current state, current exploration progress, and additional context which includes a set of questions and answers generated by GPT3.5. They demonstrate that the LLM agent can outperform state of the art in terms of obtaining unique items, travel distance, key tech tree milestones, and speed by a large margin without any specialized training or fine-tuning. The authors further highlight the lack of fine-tuning as in important characteristic of the model, with the aim of creating a starting point in developing powerful generalist agents. While the lack of specialized training is desired in a GGP agent, the abundance of available data on Minecraft indicates that the agent has been trained on this specific context. This makes the experiment unsuitable for studying LLMs performance in the role of GGP agent.

SPRING is another example of an LLM agent specialized in playing a game [21]. Wu et al. show that SPRING outperforms RL agents in playing Crafter [4] by using GPT-4. Similar to Minecraft, Crafter is an open-world survival game focused on crafting items and obtaining tools. As the authors point out, one importance distinction is that Crafter was introduced after the data collection date of GPT3.5 and GPT4, making Crafter an out-of-distribution (OOD) environment for the agent. As such, the input of the LLM agent includes the description of the Crafter environment in LaTeX format. Wu et al. employ a directed acyclic graph (DAG) representation of a set of questions including "What was the last action taken by the player?" and "Did the last player action succeed? if not, why?". By asking the questions in an order respecting the dependencies between them (shown by the edges of the DAG), they are able to create an effective chain-of-thought prompting model. Similarly, we use a chain-of-thought model to achieve better performance, even though using a DAG model directly did not result in the best performance in our case. We discuss our prompting approach further in section 4.3.

3 ENVIRONMENT

We choose a simplified implementation of the card game Slay the Spire as the testbed environment for our agent. In this section, we first introduce Slay the Spire and the core loop of its gameplay. Then, we elaborate on why Slay the Spire is particularly well-suited for our experiments. We end by introducing MiniStS, our simplified implementation of Slay the Spire, highlighting its advantages over directly interacting with the game.

3.1 Slay the Spire

Slay the Spire is a single-player deck-building roguelike card game with a strong focus on synergies between cards and strategic depth. The game features a deck-building mechanic where players collect cards while traversing a series of procedurally generated levels.

Each level in the game is a single battle against a set of enemies. The enemies follow predefined moves in a random order, and they communicate their next move, enabling the player to strategize accordingly. Initially, the player starts with all the cards in their deck shuffled in a draw pile. Each turn, the player gains 3 mana and draws a set of 5 cards from the draw pile. The player can then play any number of cards as long they have enough mana to play them. These cards range from attacking the enemies and gaining block to changing different rules of the game. When a card is played, it performs any actions described on the card and then the card is moved to a discard pile. Finally, the player ends the turn. Upon ending the turn, the player loses any remaining mana and all cards in their hand are moved to the discard pile. Finally, the enemies perform their moves, and the game enters the next turn. Enemy attacks initially reduce the player's block; when there is no block left, attacks then damage the player's HP. Any remaining block is removed from the player when entering the next turn. Whenever the draw pile becomes empty, any cards in the player's discard pile are reshuffled back into the draw pile, meaning that the player can play a card multiple times during the battle.

To discuss why Slay the Spire is an appropriate environment for our experiments, we start by introducing two example cards.

- Body Slam (cost 1): Deal damage equal to your current block.
- Barricade (cost 3): Block is not removed at the start of your turn.

As shown in the examples above, the cards which are the main mechanics of the game follow natural language descriptions, providing a wide range of potential effects. Essentially, any effect that can be articulated through a text description and adheres to the foundational rules to some extent has the potential to be added to the game. Slay the Spire specifically makes use of these natural language descriptions by defining cards that effectively change the rules. This property is not unlike the versatile rule changes during gameplay in "Baba is You" [15]. A main part of the gameplay in Slay the Spire comes from utilizing the vast space of synergies in the game, wherein the effect of two or more cards together surpasses the individual impact of each. For example, while Barricade and Body Slam are good cards individually, using both together creates a strong synergy. Barricade allows the accumulation of a substantial amount of block for defense, while Body Slam transforms this defensive strategy into a potent offensive tactic against enemies.

It's interesting to also note that this characteristic of Slay the Spire makes card generation, which is not directly attempted in this paper but is one of the motivations behind it, a challenging problem. The cards generated by the system should create effects which interact with each other directly or indirectly. Additionally, it's often hard to imagine how playing different combinations of cards (and in different orders) will affect all the other cards in the deck positively or negatively. Subsequently, any agent which plays the game must be able to draw conclusions about these positive or negative effects in order to effectively play the game.

We focus on Slay the Spire instead of other text-described card games since, as shown with the examples above, its main rules allow for complex card interactions and a heavy use of synergies. An abundance of existing mod packs introducing more card sets—including this year's "Packmaster Mod" created by the community which adds more than 700 new cards to the game—show how well the foundation of the game can provide a diverse natural testing ground for game designers and PCG systems.

Second, Slay the Spire is a single player game, which reduces the complexity of evaluating agents since they are always evaluated against an enemy with a predetermined set of moves. In other words, the strength of an agent can be evaluated independently from any opponent agents. This removes many inherent complexities in deck-building and game meta research done in works such as [2]. Additionally, while the game is not deterministic because of deck shuffling and randomness in the order of enemy moves, the absence of an opponent with a diverse array of possible cards in her hand narrows the range of potential outcomes for each move.

Finally, it's important to note that in contrast with many other card games which are focused only on winning the game, in Slay the Spire the remaining health points at the end of battle is important. This is due to the fact that remaining HP is carried over to the next fight. For example, when a battle is won with low HP at the end, the player is likely to lose in the next battle, so the final HP more accurately measures the success of the agent in playing the game. Because of this property, we use remaining health at the end of the battle as one of the main metrics to evaluate our agents. Note that the final player health is positive if and only if the battle is won, so it can better distinguish between different win scenarios.

3.2 MiniStS

Of course, while the natural language descriptions on cards make it possible to define any text-describable effect, in practice the engine must be able to support the implementation of a card for it to be added to the game. To this end, we introduce *MiniStS*[1] , a simplified implementation of Slay the Spire designed specifically to allow for addition of new cards to the game. Slay the Spire provides sufficient tools to develop mods for adding new card packs as well as to connect to a game playing agent[2]. However, MiniStS allows for easier modifications of the available card set in the game as well as faster execution compared to connecting to the game through mods.

In Slay the Spire, many of the specialized rules are shown as status effect on the playable character. For example, the card Barricade applies the status effect *Barricade* on the character for the rest of battle. We follow the same format since it allows for the definition of different effects and, more importantly, it lets the agents know how the rules have changed in the current state of the game without requiring a history of cards played during the battle. To make this possible, we use a set of base classes defining different actions (e.g. gain block, deal damage, apply status effect, etc.), enemies, and status effects. The cards are then defined by specifying the list of actions performed upon playing them. These can be decorated by functions such as "And" (to add another action at the end) and "To" (to describe how the target of this action is selected).

Finally, an event system makes all this possible by providing a way of registering on different events and broadcasting them at certain points. For example, a status effect can register for the event related to end of turn, defining a function which is called at the end of every turn and changes the state of the game in some way.

It's worth mentioning that for our experiments, we use 80 as the starting health value for the player, similar to the starting health for IronClad, the first of the four playable characters in Slay the Spire. We also use a simple enemy with two possible moves: deal 22 damage, or block for 10 and deal 10 damage. The moves are chosen based on the existing moves for enemies, but with higher values for damage and block to better differentiate between a successful and unsuccessful strategy. All of the experiments are done by simulating a battle against one enemy of this type.

4 AGENTS

We define a game playing agent as an abstract module expected to answer two kinds of questions. First, given the state of the game and a list of moves, the agent should return a single move to play, such as "Play card X from your hand" or "End turn". The list of options only includes valid options with considerations for conditions such as having enough mana points to play the card. Second, since some of the cards require one or more targets, given a list of possible targets the agent should return the selected target. The target can be an enemy (e.g. deal 10 damage) or even a card (e.g. discard a card

[1] Our Python implementation of MiniStS can be found at https://github.com/iambb5445/MiniSTS

[2] For example, we were able to connect our LLM agent to Slay the Spire through use of *spirecomm* mod and *communication mod*. But first, even with the addition of mods to speed up the animations, this is much slower compared to the headless execution of MiniStS. And second, since the game has many more components such as relics, events, procedurally generated map, etc, this makes it hard to evaluate our agents on reasoning capabilities alone.

from your hand). Based on this definition, we have implemented three different types of agent so as to compare their performance.

4.1 Random Agent

The random agent is our baseline to not only compare randomized play against the carefully selected moves of other agents, but also to prevent unfair scenarios. In other words, if the deck of cards allows for a fairly good strategy to be randomly selected while, for example, the backtrack agent gets stuck in a local optimum, the random agent might perform better than the backtrack agent. However, since this does not happen in the provided scenarios, we can show that smart (not necessarily optimal) play is required to perform well in every scenario.

4.2 Backtrack Agent

To evaluate the LLM agent, we compare it with a backtrack agent. Since our main goal is to assess an agent in performing as a GGP agent, ideally we want our agent to show characteristics of generalized play without training specifically for each scenario. Although training from scratch for every given ruleset can be useful in some cases [5, 17], our focus here is on the development of generalized agents specialized in rapid simulation of widely different scenarios. Inspired by the 1 turn look-ahead agent used in Chaos Cards [2], we create a *n-step* look-ahead backtrack agent. The backtrack agent decides the next best move by simulating all possible combinations of *n* moves ahead and choosing the move which maximizes the expected score on the final state of the game after performing the next best *n* moves. The score for any game state is based on whether the game has ended, and the amount of remaining health for the player and each enemy. Note that the predictions of future states are not always completely valid, since some randomness exists in drawing new cards and shuffling the deck. This is important since we don't want our agent to have perfect information about the future, as a human player also lacks this information.

An important characteristic of this agent is that it is not trained in any way on what the moves are and only draws conclusions based on the outcome of performing them. This means that the agent is able to respond to any changes in the game environment or ruleset without any modifications.

4.3 LLM Agent

Given the state of the game and a set of options to choose from, the LLM agent creates a prompt to include all the necessary information. As shown in Fig. 1, this prompt is a combination of three main components. First, the game context includes the global information about the game, such as information on drawing cards, spending mana, status effects, and the description of existing cards in the deck. Even though most scenarios include completely new cards, we anonymize the name of cards to minimize the risk of our agent playing the cards based on available training data about Slay the Spire. We replace the name of each card with a randomly generated string of 6 characters. Interestingly, this improved the performance of the LLM agents. We speculate that this is due to the LLM agent mostly relying on the familiar card names instead of newly introduced cards when the names are not anonymized.

In addition to the game context, we also include the game state in our prompt. The game state includes information such as mana and turn number as well as hp, block and status effects of the player and each enemy. In Slay the Spire, the intention of each enemy (e.g. attack for 10, block, etc.) for this turn is visualized in the UI of the game to create gameplay focused on careful planning and strategy. Similarly, we include the intention data in our prompt to give the agent information similar to what is given to a human player.

Finally, we specify what the LLM is expected to return. These three components result in a prompt with around 1,000 tokens for each move. The prompt is sent to the LLM and the output is interpreted based on the request. In case the output does not have the required move index or the move index is out of range, the response is considered invalid and the agent retries sending the prompt to the model. We discuss the rate of invalid responses for different models in section 4.3.2.

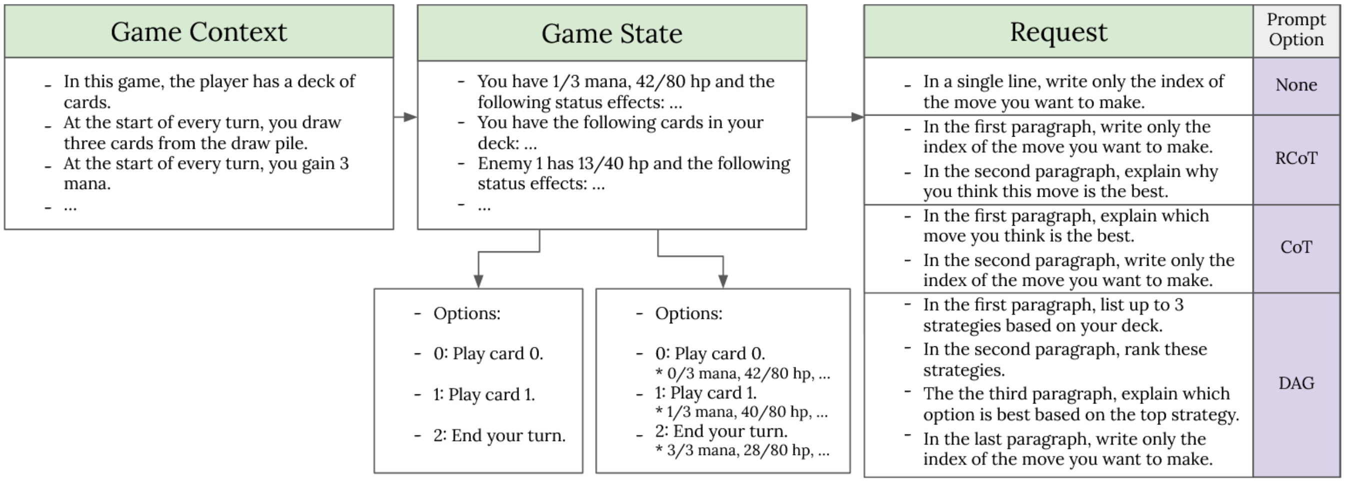
4.3.1 *Prompt Options.*

The request component of the prompt can ask for different information in the output. The type and order of the information can significantly affect the performance of our agent. We use the GPT3.5 model [9] in the starter deck scenario (described in section 5.1) to compare four different options. The final results of this experiment can be seen in Fig. 2.

(1) None: This option only asks the underlying LLM model for the index of the best move. We use this option as a baseline to compare how asking for additional information affects the performance of the agent. Without using any prompt options, the LLM agent performs as poorly as a random agent, demonstrating the importance of prompt approach. Note that the rest of the prompt (i.e., game context and game state components) are identical across prompt options.

(2) Chain-of-Thought (CoT): We use a chain of thought model to push the agent toward logical answers by asking for explanations. Note that some works in the literature [20] use examples of step-by-step thinking to motivate the agent to generate logical answers, but in our case this can prove difficult since any example of the prompt will take around 1,000 tokens to include. Additionally, examples of how the agent should react in a certain scenario can bias the agent since there are many different possibilities for the current state of the game, which require different types of strategies and actions. So, for this option, we ask the agent first for an explanation of the best move, and then for the index of that move. As shown in Fig. 2, asking for an explanation significantly improves the quality of the output.

(3) Reverse Chain-of-Thought (RCoT): As part of our experiments, we asked the agent to give explanations about why it chose a certain move. We kept this option in this experiment for the sake of comparison. Note that the only difference between RCoT and CoT options is the order in which we expect the agent to return explanation and best move. We expected this option to perform worse than CoT. Interestingly, we see that there is some performance gain over Random and None when RCoT is used. But, as expected, this gain is much less than regular CoT. We include these results here to highlight the importance of the order in which explanation and answer



Game Context

- In this game, the player has a deck of cards.
- At the start of every turn, you draw three cards from the draw pile.
- At the start of every turn, you gain 3 mana.
- ...

Game State

- You have 1/3 mana, 42/80 hp and the following status effects: ...
- You have the following cards in your deck: ...
- Enemy 1 has 13/40 hp and the following status effects: ...
- ...

- Options:
- 0: Play card 0.
- 1: Play card 1.
- 2: End your turn.

- Options:
- 0: Play card 0.
  * 0/3 mana, 42/80 hp, ...
- 1: Play card 1.
  * 1/3 mana, 40/80 hp, ...
- 2: End your turn.
  * 3/3 mana, 28/80 hp, ...

Request	Prompt Option
- In a single line, write only the index of the move you want to make.	None
- In the first paragraph, write only the index of the move you want to make. - In the second paragraph, explain why you think this move is the best.	RCoT
- In the first paragraph, explain which move you think is the best. - In the second paragraph, write only the index of the move you want to make.	CoT
- In the first paragraph, list up to 3 strategies based on your deck. - In the second paragraph, rank these strategies. - The the third paragraph, explain which option is best based on the top strategy. - In the last paragraph, write only the index of the move you want to make.	DAG

Figure 1: Elements of the LLM prompt. The game context encapsulates the overarching rules of the game, while game state includes the current state of the game and available options. The options can be considered with or without mentioning their outcome. Finally, the request asks for specific output based on the selected prompt option (e.g. chain-of-thought, etc.)

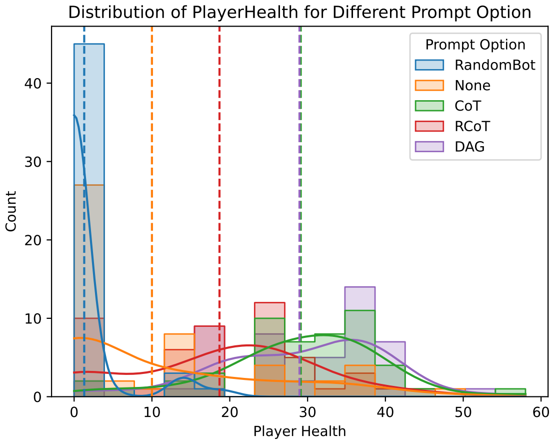


Figure 2: Comparison between the performance of different prompt options over 50 simulations per agent. Each agent is represented by a histogram of final player health. The mean values are shown with vertical lines. The simulations use a starter deck, as discussed in section 5.1.

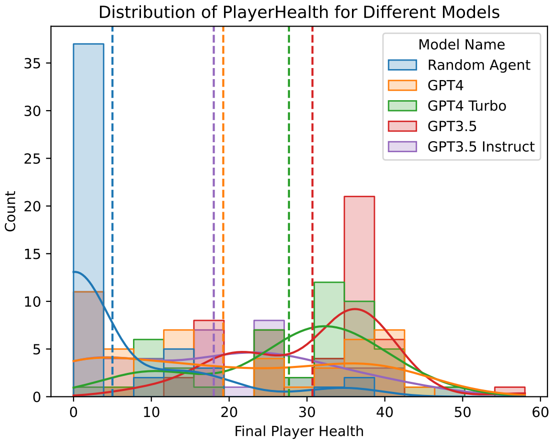


Figure 3: Comparison between the performance of different models over 50 simulations per agent. Each agent is represented by a histogram of final player health. The simulations with a value of zero are associated with a lose condition. The simulations use a starter deck, as discussed in section 5.1.

are generated by the model. This is expected since the model can hallucinate an explanation for any previous move.

(4) DAG: Inspired by the SPRING Minecraft agent [21], this approach asks for up to three possible strategies, a ranking of them, the best move to take based on the best strategy and current state, and finally the index of this move. While this improves the quality of the play in some cases, the average value is not improved as compared to the CoT option. Furthermore, in our case we found that asking for this information from the model results in many invalid responses where the model does not return the index of the best move in the final paragraph, whereas the other options rarely result in an invalid move. Because of this, we only use the None and CoT options for the rest of our experiments.

As discussed above, we use the None and CoT options for our main experiments. The complete prompt as well as some example requests and responses can be found in the previously mentioned MiniStS GitHub repository.

4.3.2 Model Comparison. We use OpenAI's GPT models in our experiments. To find out which one of these models offers the best

Table 1: Comparison between the execution time of different models. Time values are in seconds. The execution time is the total time over 50 simulations run in parallel divided by the number of simulations, and does not represent the length of one simulation.

	Avg. Execution Time per Simulation in Parallel	Avg. Response Time	Invalid Response Percentage
GPT3.5	9.29	1.58	1.38
GPT4	14.75	5.76	0.00
GPT4 Turbo	5.72	11.75	0.75
GPT3.5 Instruct	55.48	0.62	80.46

performance, we compare them in the starter deck scenario. The results of this experiment are shown in Fig. 3. GPT3.5 not only achieves better results in terms of quality of play in the starter deck scenario, but is also significantly faster than the second best performing model, GPT4. This is both because of GPT4's more restrictive rate-limits imposed by OpenAI and the response time of the model. Because of this, we use GPT3.5 for our main experiments.

Table 1 summarizes the execution times of each model. Note that the average execution time per simulation is calculated over a parallel run of 50 scenarios on 50 threads, and does not represent the length of a single simulation. Surprisingly, on average GPT4 responds faster than GPT4 Turbo, but because of the aforementioned rate limits it is overall slower than both GPT4 Turbo and GPT3.5. On the other hand, GPT3.5 Instruct is faster than all the other models, but it has a significantly higher invalid response rate of about 80%. A response is considered invalid if it doesn't follow the required formatting (i.e. index of the best move in the last prompt) or if the index of the best move is not one of the valid options (i.e. non-integer or out of range). Because of this struggle with following the correct output format, simulations with GPT3.5 Instruct take significantly longer compared to the other models. It's also worth mentioning that the average execution time can be deceiving, since a better player can survive the battle longer and get a worse execution time.

4.3.3 History. Finally, we compare how including the history of past moves can affect the performance of our agent. For this experiment, we compare three different approaches. First, for the stateless *No History* agent, the prompt includes only the game context, game state and request as visualized in Fig. 1. We compare this with the *Past 5 Moves* agent which keeps the history of last 5 moves. The prompt for this agent includes game context, the last five game states, requests and responses, ending with the current state and request. For an alternative history option (*Options Outcome* agent), we include the outcome of each option in our game state. For each available move, we add the expected stats for player and enemies, such as their HP and status effects. As shown in Fig. 4, both adding the outcome and adding the past 5 moves result in worse performance from the agent. This is most likely due to the inability of GPT3.5 to attend to all the information in its context window, to the point that the additional information causes a decrease in the performance. In contrast, we noticed a performance gain when experimenting with the same three options on GPT4 Turbo.

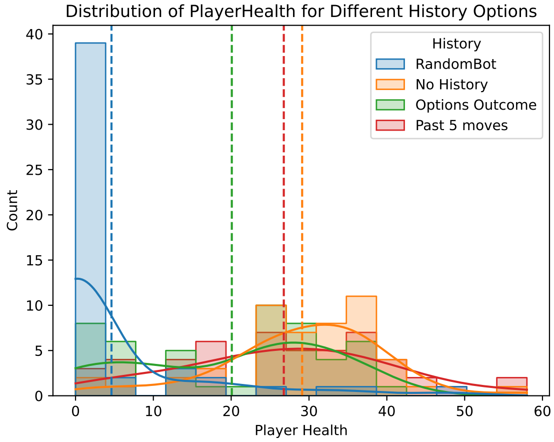


Figure 4: Comparison between the performance of different history options over 50 simulations per agent. Each agent is represented by a histogram of final player health. The mean values are shown with vertical lines. The simulations use a starter deck, as discussed in section 5.1.

5 EXPERIMENTS

To assess how well the LLM agent can perform in understanding the game mechanics and performing appropriate moves in the game, we experiment with 4 different scenarios. Since our main motivation is to evaluate different cards in the game, each scenario uses a different deck of cards. For all of the scenarios, we use player health at the end of the game as one of our main metrics, as discussed in section 3. As for our agents, we compare 4 agents for every scenario: random agent, backtrack agent with depth 3, and LLM agents with prompt options None and CoT. Based on our experiments in section 4.3, we use GPT3.5 [9] as our underlying LLM model, and we use this model in a state-less manner without any history mechanism. The rest of this section explains each scenario and analyzes the performance of different agents in that specific context. Table 2 summarizes the results for the first four experiments.

5.1 Starter Deck Scenario

The first scenario uses the starter deck for one of the characters in the game. This deck is used at the start of the game for the first battle. While this deck does not have strong synergies, it acts as a starting point and blank slate for any future deck-building. We use this deck to evaluate our agents on how well they can play the basics of the game without a heavy focus on strategy. The deck includes the following cards:

- Strike (cost 1): Deal 6 damage.
- Defend (cost 1): Gain 5 block.
- Bash (cost 2): Deal 8 damage and apply 2 *Vulnerable.* (2 *Vulnerable*: receive 50% more damage for 2 turns)

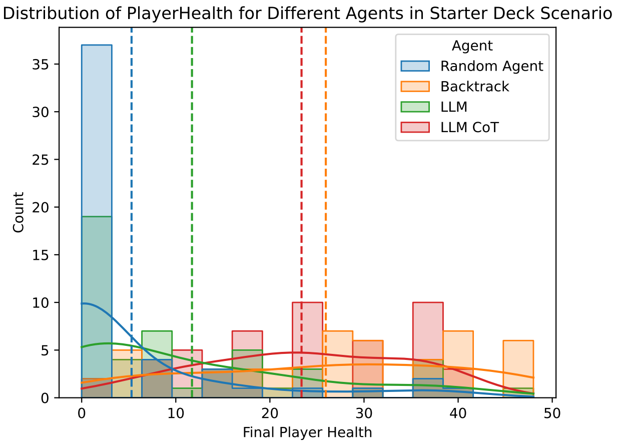


Figure 5: Comparison between the performance of different agents over 50 simulations per agent in the starter deck scenario. Each agent is represented by a histogram of final player health. The mean values are shown with vertical lines.

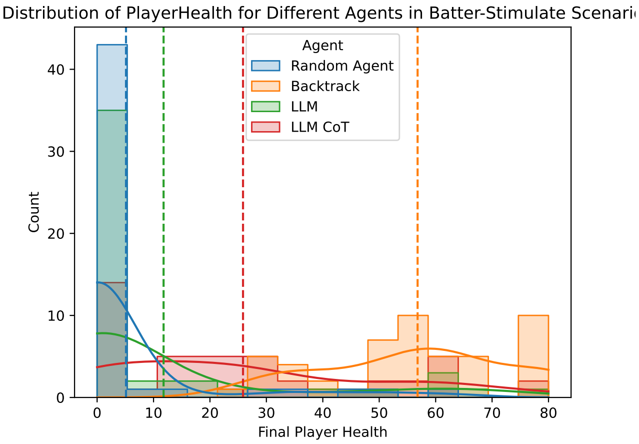


Figure 6: Comparison between the performance of different agents over 50 simulations per agent in the Batter-Stimulate scenario. Each agent is represented by a histogram of final player health. The mean values are shown with vertical lines.

The starter deck in the game consists of 5 Strike cards, 4 Defend cards, and 1 Bash card. As shown in Fig. 5, while the LLM agent with CoT plays intelligently and wins the game (whereas the random agent mostly loses), the backtrack agents perform better in this scenario. Since the starter deck only includes basic attack and defense cards, the best strategy is to defend for incoming damage and attack for the rest of your mana. The backtrack agent is better at predicting remaining health after playing combinations of attack and defend cards, and can optimize play to maximize player health.

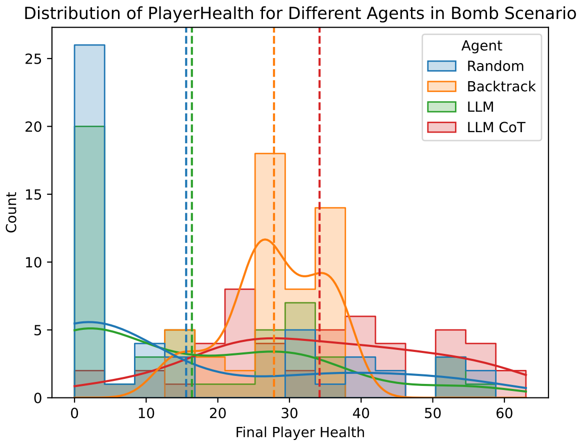


Figure 7: Comparison between the performance of different agents over 50 simulations per agent in the Bomb scenario. Each agent is represented by a histogram of final player health. The mean values are shown with vertical lines.

5.2 Batter-Stimulate Scenario

The second scenario uses a deck with a short-term synergy between cards, that is, a synergy which expresses in short-term play, such as playing two cards one after the other. We introduce two new cards to create a deck with a short-term synergy:

- Batter (cost 1): Deal 0 damage 10 times.
- Stimulate (cost 1): Gain 4 *Vigor*. (4 *Vigor*: apply 4 additional damage on the next card you play.)

For this scenario, we use 5 Strike and 4 Defend cards (same as the starter deck), along with 1 Batter and 1 Stimulate card. As shown in Fig. 6, backtrack agents perform much better in this scenario. The reason is that this scenario is designed to greatly favor using Batter and Stimulate cards together. Using Batter after Stimulate without playing any other attack cards in between will result in 40 damage, almost killing the enemy (with a health of 44). While the backtrack agent is able to see this exact play every time, the LLM agent uses this strategy only half the time. As a result, while the backtrack agent improves its performance as compared to the starter deck scenario, the LLM agent with CoT has only a slight improvement.

5.3 Bomb Scenario

The third scenario introduces a long-term strategy. One of the existing cards in the game, Bomb, is defined as follows:

- Bomb (cost 2): At the end of 3 turns, deal 40 damage to all enemies.

The deck used in this scenario consists of 5 Strike and 4 Defend cards, and 1 Bomb card. Note that while this card exists in the game, recall that we anonymize the name of this card (and other cards in the game) to prevent the LLM from exploiting pre-training knowledge on when to use this card. Players are motivated to use this card when incoming damage is low and the player is expected to survive for 3 more turns. As expected, the backtrack agent is

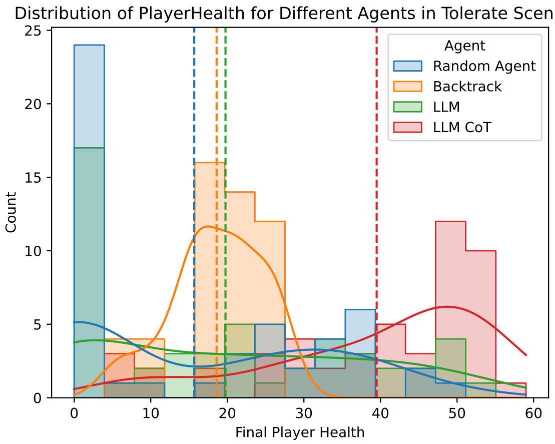


Figure 8: Comparison between the performance of different agents over 50 simulations per agent in the Tolerate scenario. Each agent is represented by a histogram of final player health. The mean values are shown with vertical lines.

not able to see far enough in the future to understand the effect of this card, while the LLM agent is perfectly capable of playing this card in appropriate scenarios. Fig. 7 shows the distribution of final player health values over 50 simulations.

5.4 Tolerate Scenario

The Tolerate scenario involves changing rules of the game. Inspired by existing cards such as Deva Form (which gives the player 1 energy at the start of every turn and increases this gain by 1 every turn) or Demon Form (which gives the player 2 extra damage on your attacks every turn and increases this gain by 2 every turn), we introduce:

- Tolerate (cost 3): Gain 1 block every turn and increase this gain by 2. [Power card, which means that it will be removed from the game after playing it.]

The deck includes 1 Strike, 3 Defend and 1 Tolerate card. Since the deck consists of 5 cards, the player draws all five every turn. This gives the player the following choice: Either block for 15 (play 3 Defend cards), attack and block for 10 (since playing an attack card leaves the player with insufficient mana to play Tolerate), or play Tolerate. While initially playing Tolerate and accepting the incoming damage does not sound beneficial, the fact that only 1 attack card is in the deck means that the player cannot win the game for many turns to come. Because of this, it's important to play Tolerate by sacrificing a turn, with the goal of surviving long enough for the Tolerate effect to become helpful. This gives the player enough block late in the game to focus entirely on offense. While the LLM agent understands this, the backtrack agent does not see enough merit in the near future to play Tolerate.

Finally, it's important to note that in this scenario performing well comes not only from playing Tolerate, but more importantly in understanding that Tolerate should be played as fast as possible. Fig. 9 shows the distribution of the turn in which Tolerate was played by each agent. As was discussed earlier, this scenario is designed to play for many turns (since the only available attack card cannot be used more than once a turn). As a result, the Random agent ends up playing Tolerate at some point in the game. However, as seen in the final player health, it often ends up losing the game since it plays Tolerate too late to be helpful. In contrast, the LLM agent with CoT ends up using Tolerate within the first 2 turns more than half the time.

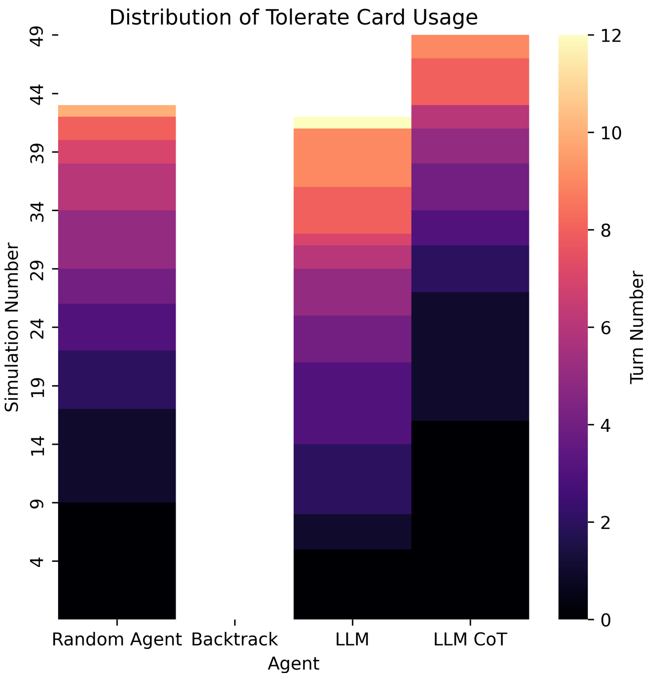


Figure 9: The distribution of which turn Tolerate was played (if at all). While agents such as random agent and the regular LLM agent also frequently use Tolerate, they frequently lose the game since the card is often played too late.

5.5 Harm Scenario

For our final scenario, we test the agents in understanding when *not to* play a card. To do so, we create a simple scenario in which the deck consists of 5 Strike cards, 4 Defend cards, and 1 Harm card defined as follows:

- Harm (cost 1): Deal x damage.

The Harm card has a similar definition to strike, but with a custom amount of damage. We test this scenario with different values for x to see whether or not the agent continues to use the card when the card is not useful anymore. We calculate the percentage of turns in which the agent plays this card out of all the turns that the agent could have played the card, i.e. the card was drawn from the draw pile. For example, if the agent plays the game for 10 turns, in 4 of which Harm is drawn and available to play, and the agent plays Harm once, this percentage is 25%. We run 50 simulations of this scenario for each value of x: 1, 5 and 10. Table3 summarizes the results.

Table 2: The average value of final player health over 50 simulation per scenario and agent.

Scenario	Random Agent	Backtrack Agent	LLM Agent	LLM Agent with CoT
Starter Deck	5.3	25.94	11.72	23.36
Batter-Stimulate	5.12	56.8	11.78	25.68
Bomb	7.7	17.2	10.42	34.74
Tolerate	15.68	18.6	19.76	39.5

Table 3: The percentage of turns Harm card was played out of all the turns it could have been played for each agent over 50 simulations each.

	Harm x=1	Harm x=5	Harm x=10
Random Agent	37	33	41
Backtrack	23	97	99
LLM CoT	42	73	77

The results show that both the backtrack and the LLM agent demonstrate an understanding of the value of Harm in each scenario. Note that a regular strike does 6 damage, so x=10 can be considered a good card, x=1 is a weak card, and x=5 is close to an average starter card, and the agents react to all three cases. The backtrack agent more strongly reacts to the x=1 scenario, since it can better predict the outcome of playing this card compared to for example playing a Defend, but all agents play the card less frequently as the amount of damage the card offers decreases.

6 CONCLUSION

In the Ludi system by Browne and Maire, GGP is described as "software systems for playing a range of games well rather than any one particular game expertly." In this paper we propose the possibility of using an LLM agent for the purpose of general game playing, observing that LLMs exhibit non-optimized but highly generalizable play. To analyze the ability of LLMs to perform general game playing tasks, we used our implementation of Slay the Spire, MiniStS, which permits the definition of new cards and testing of different scenarios. Through this setup, we demonstrate that when paired with a chain-of-thought technique, LLMs can play the game in an acceptable manner compared to other agents, and outperform the other agents in cases where the agent must be able to reason further than the next few turns. Our research shows a promising path for using LLMs as generalized models for playing games in a dynamic environment, enabling the possibility of using them to evaluate the open-ended procedural generation of novel rules in a variety of game scenarios.

REFERENCES

[1] Cameron Browne and Frederic Maire. 2010. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (2010), 1–16. https://doi.org/10.1109/TCIAIG.2010.2041928

[2] Tiannan Chen and Stephen Guy. 2020. Chaos Cards: Creating Novel Digital Card Games through Grammatical Content Generation and Meta-Based Card Evaluation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16, 1 (Oct. 2020), 196–202. https://doi.org/10.1609/aiide.v16i1.7430

[3] Mega Crit Games. 2019. *Slay the Spire*.

[4] Danijar Hafner. 2022. Benchmarking the Spectrum of Agent Capabilities. arXiv:2109.06780 [cs.AI]

[5] Johor Jara Gonzalez, Seth Cooper, and Matthew Guzdial. 2023. Mechanic Maker 2.0: Reinforcement Learning for Evaluating Generated Rules. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 19, 1 (Oct. 2023), 266–275. https://doi.org/10.1609/aiide.v19i1.27522

[6] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent Predictor Networks for Code Generation. arXiv:1603.06744 [cs.CL]

[7] M Milewicz. 2016. RoboRosewater. https://twitter.com/roborosewater

[8] David Noever, Matt Ciolino, and Josh Kalin. 2020. The Chess Transformer: Mastering Play using Generative Language Models. arXiv:2008.04057 [cs.AI]

[9] OpenAI. 2023. ChatGPT (June 13 version). https://chat.openai.com/chat. Large language model.

[10] Barney Pell. 1992. METAGAME: A new challenge for games and learning. (1992).

[11] Barney Pell. 1996. A STRATEGIC METAGAME PLAYER FOR GENERAL CHESS-LIKE GAMES. *Computational intelligence* 12, 1 (1996), 177–198.

[12] Adam M. Smith and Michael Mateas. 2010. Variations Forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. 273–280. https://doi.org/10.1109/ITW.2010.5593343

[13] Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. 2023. MarioGPT: Open-Ended Text2Level Generation through Large Language Models. arXiv:2302.05981 [cs.AI]

[14] Adam Summerville and Michael Mateas. 2021. Mystical Tutor: A Magic: The Gathering Design Assistant via Denoising Sequence-to-Sequence Learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 12, 1 (Jun. 2021), 86–92. https://doi.org/10.1609/aiide.v12i1.12851

[15] Arvi Teikari. 2019. *Baba is You*.

[16] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level Generation Through Large Language Models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games* (Lisbon, Portugal) *(FDG '23)*. Association for Computing Machinery, New York, NY, USA, Article 70, 8 pages. https://doi.org/10.1145/3582437.3587211

[17] Julian Togelius and Jurgen Schmidhuber. 2008. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*. 111–118. https://doi.org/10.1109/CIG.2008.5035629

[18] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

[19] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291 [cs.AI]

[20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[21] Yue Wu, Shrimai Prabhumoye, So Yeon Min, Yonatan Bisk, Ruslan Salakhutdinov, Amos Azaria, Tom Mitchell, and Yuanzhi Li. 2023. SPRING: GPT-4 Out-performs RL Algorithms by Studying Papers and Reasoning. arXiv:2305.15486 [cs.AI]