MASTER THESIS

Alexander Chubar

Generating game content with generative language models

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Vojtěch Černý
Study programme: Visual Computing and Game Development

Prague 2024

First of all, I would like to thank my supervisor, Vojtěch Černý, not only for guidance throughout the thesis and valuable advice but also for patience and psychological support. Equally important, I would like to express my gratitude to all the teachers from the Game Development study program I had a chance to learn from. I would also like to thank my dear wife, family and friends, who have been supportive, helpful, and patient throughout my studies.

Title: Generating game content with generative language models

Author: Alexander Chubar

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Vojtěch Černý, Department of Software and Computer Science Education

Abstract: The objective of this thesis is to study the capabilities of Large Language Models (LLMs) in Procedural Content Generation for games (PCG). We would like to explore this from multiple perspectives and understand how we can combine PCG techniques with approaches for applying LLMs across various fields. To achieve our goal, we create a game in the Japanese Role-Playing Game genre that utilises structured content generated by LLMs as input and uses it to provide a diverse and varied game experience. Additionally, we will analyse the generated content using several metrics. This analysis will help us to evaluate the effectiveness of LLMs and suggest possible further advancement in this rapidly emerging field.

Keywords: Procedural Content Generation, Generative Language Model, Computer Games, Large Language Model, Generative Pretrained Transformer

Contents

Introduction

The creation of games is an intellectual and creative process that requires endeavours from people with diverse skills. Fortunately, resources can be conserved by employing techniques that allow developers to automate certain aspects of game development, such as Procedural Content Generation (PCG) [1]. PCG is the process of using different algorithms to generate game content according to predetermined rules. It can be applied to various types of games and game content. However, we will be exploring its usage applied specifically to video games.

PCG has been used in game development for decades. One of the most iconic and genre-forming examples is the game Rogue (1980) [2]. Rogue inspired an entire genre of games known as roguelikes. Along with permadeath (after dying, players need to start the game from scratch), non-modality (every action should be available at any point of the game), and a few other characteristics, procedurally generating dungeons have been one of the pillars of the roguelikes genre [3]. It remains very popular today but has undergone many changes over time and is still one of the primary genres frequently referenced as an example in discussions about PCG.

PCG has many different uses in game development. It can be used to create expansive environments, items, enemies, textures, music, narratives, etc. No Man's Sky (2016) [4] is a good example of generating an endless, distinct world with diverse terrains and ecosystems. In games like Diablo 2 (2000) [5], PCG is used to create countless unique randomised armours, weapons, and enemies, which provides diverse gameplay and replayability. Dwarf Fortress (2006), discussed in chapter 1.3.2, uses PCG to create complex stories with randomly generated characters and environments, which enhances replayability and player engagement. There are a lot of other examples at different scales. Even the initially randomised launch angle of the ball in Pong (1972) [6] can be considered a form of procedural generation. Early examples like Pong show us how simple PCG elements can add variety and replayability to a game.

While there are many different approaches to PCG, in this diploma thesis, we will introduce generating game content with large-scale generative language models, namely, GPT, which is discussed in Chapter 1.2.2. Even though LLMs include various architectures and families of models, for this thesis, we will refer to large generative language models of GPT type as Large Language Models (LLMs). Since LLMs have emerged and developed relatively recently, using them for PCG is also a fresh and actively studied field.

Nowadays, the scope of LLMs is extended by the emergence of multimodal systems that can also be applied to audio and visual content. However, natural language application still stays the most popular one. There are a few undisputable advantages offered by LLMs that might make them useful for PCG and are worth studying in the scope of this diploma thesis. LLMs are trained on huge amounts of data, which makes them a good source of rich and diverse narratives, dialogues, and storylines. Also, modern LLM APIs are user-friendly and affordable, using them doesn't require specific skills or extensive resources. Therefore, it is accessible for indie developers and small studios. There is a lot of freely accessible information on techniques and approaches to LLMs which can be applied to PCG as well. Thus, this thesis will primarily focus on text generation applications and using multimodal LLMs will stay out of the scope for this work.

The result of the diploma thesis is a game of the Japanese Role-Playing Game (JRPG) genre made in Unity and an application that interacts with a language model to generate a Game World that is used as an input for this game. The data, scope and format of this Game World are restricted by the existing game systems and content.

This diploma thesis is structured as follows. Chapter 1 covers the theoretical background regarding the JRPG genre relevant to this thesis, PCG usage cases and techniques, and the theoretical foundation of LLMs. In Chapter 2, we will overview the game created for this thesis, called GPT JRPG. We will discuss the structure of the game, how it applies the idea of generating content with LLMs, and how it will use the generated content. Chapter 3 begins with a discussion of prompt engineering and its fundamental techniques. After that, we will take a closer look at how the interaction with GPT is performed for this thesis and discuss ways various issues were addressed. In Chapter 4, we will discuss and evaluate the results of the thesis. In Chapter 5, we will speculate on potential alternative approaches for this thesis and suggest how this topic can be explored in future.

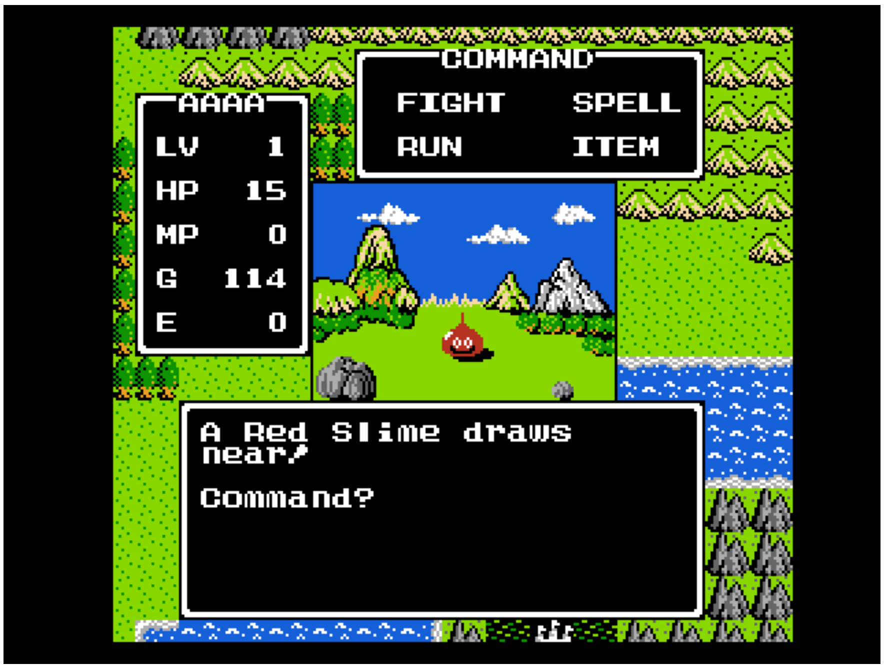
1 Background

1.1 JRPG

In this section, we will discuss a Japanese Role-Playing Game (JRPG) genre chosen for the game created for the thesis experiment. The definition behind the term is considered to be broad and subjective. On the one hand, this term might refer to any game of the RPG genre produced in Japan. On the other hand, it can be considered an RPG sub-genre with specific features. For example:

- Manga-style art.
- Recognisable UI appearance.
- Linear narrative, focused on the main story.
- Predefined characters.
- Party and turn-based combat systems.
- Level progression system.

These features are not very strict and have been transformed over the years of JRPG development [7]. For this thesis, we will use the second definition of the JRPG genre and consider these features as a foundation for the game we create. One of the most iconic worldwide successful JRPG examples is Dragon Quest(1986), shown in the figure 1.1.



A Red Slime draws near!

Command?

Figure 1.1 Combat scene from Dragon Warrior, English localisation of the classic JRPG Dragon Quest.
Source: Custom screenshot from the game made by Spike Chunsoft

It has typical features like a turn-based battle system and, in this case, a first-person view. It also has a multi-level world shown in figure 1.2, a well-known JRPG feature. There is usually a World Map that contains objects representing smaller-scale locations (Local maps). On these Local Maps, players can face enemies who might engage in combat that is happening in a distinct location (Combat Area). Most of the interactions within the game are in textual format, which is typical, especially for classic (late 20th-century JRPGs). In later games of this series, many other typical JRPG features were introduced, like playing as multiple characters and being able to form a party with existing in-game characters.

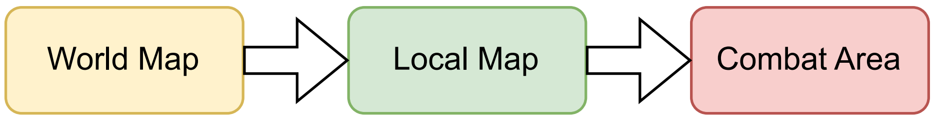


Figure 1.2 Typical world structure for a classic JRPG game.

When creating a JRPG, there are several main components indie and solo developers should be focused on:

- **Story.** Classical JRPGs are usually story-driven. The plot should be clear and concise. For simplicity, there should be a main idea around which the plot revolves, like love, sacrifice, betrayal or good forces fighting against evil, etc.
- **Characters.** Protagonists and antagonists, with their supportive characters, should be woven into the storyline. Each character might have different personality traits, which are usually quite clearly reflected in dialogues with them. Their personality is usually defined by their backstory and their position in the fiction world.
- **Combat System.** While classical JRPGs usually feature the turn-based combat system, it can be real-time or even be represented by something like a cardboard minigame [8] or as an auto battler [9].
- **RPG System.** Due to the genre foundation, there should be some progression system. This can be represented by crafting, levelling, collecting items, or group managing systems.

There is a lot more to it, like being consistent with the visual art and sounds, implementing systems that are dependent on the player's choice, etc.

Today, JRPG might be considered an outdated term. However, some indie games are being developed today that capture the essence of the JRPG genre, like the award-winning Sea of Stars (2023) [10]. It was specifically chosen to make a game with some qualifying features of JRPGs of the 80s and 90s era because it is relatively easy to produce using modern game engines. Also, it usually uses quite a large amount of textual content compared to other genres, which is a perfect case to explore the usage of LLMs for procedural game content generation.

1.2 Large Language Models

Large Language Models (LLMs) represent a class of powerful artificial intelligence models that work with natural language and are built using machine learning techniques. They are defined by their large scale (Modern LLMs are normally trained on trillions of tokens and have a large number of parameters, 200 billions in the case of GPT-4o [11]), specific architecture (toady LLMs are mainly built on a transformer architecture), and diverse capabilities (LLMs usually can handle a wide variety of tasks and serve a diverse set of applications).

LLMs have been spotlighted over the last few years and have a vast amount of known use cases. This rapid development of LLMs started in 2017 with the Google Brain team's breakthrough paper "Attention is all you need" [12]. In this paper, the Transformer architecture was first discussed with the attention mechanism that powers this architecture. It boosted the development of the Natural Language Processing field, creating subsequent models that exploit the main idea of the Transformer architecture. Nowadays, the most well-known and capable LLMs use some form of the Transformer architecture as a foundation, we will discuss some of them in the following sections.

The field of applications for LLMs is quite broad. However, their primary application revolves around understanding and generating natural language. It can be used to write technical documentation, assist people in learning, academic and creative writing, legal consulting, and cancer prediction. It is already used in different fields to generate data, like educational content, marketing and copywriting, and, of course, to generate responses in chatbots. [13] [14] [15]

We should mention that following a strict structure is considered to be a challenging task for language models. However, the most modern models have features that can help mitigate this issue (like JSON mode for OpenAI API GPT models [16]). Overall, LLMs are a promising choice to be tested for procedural content generation. In this thesis, we would like to explore how LLMs can be applied as PCG for video games, namely generating characters, storylines, quests, and so on, as well as their possibilities and limitations in this regard. Successful use of LLMs in PCG can provide us with an instrument that can help developers save time and resources while diversifying in-game content for players.

Even though in the thesis, we use mainly generative LLMs and their openly available APIs without training a model from scratch or changing their internal architecture, we will mention some other language model architectures to provide a superficial background on the field of language models.

1.2.1 Theory behind LLMs

Language models are neural networks that are trained on natural language data. Their primary goal is to handle natural language understanding or generation tasks that are acquired during the training.

A neural network is a computational model that was inspired by the way how a human brain works. A neural network is composed of multiple neurons organised in layers. The structure of neural network is shown in figure 1.3. In order for a neural network to learn how to perform a specific task, the data should be provided showcasing how from a specific input we can come to a specific result. Then, the data is used to train a neural network. The training involves calculating a loss function (a measure of error during the task solving), and the mechanism that updates weights and biases in neurons (usually, a backpropagation algorithm is used). During the training process, the backpropagation algorithm tweaks the neural network's weights in the right direction. Normally, it takes several passes through the data to pick up the pattern the data might have and to be able to solve the task during the inference.

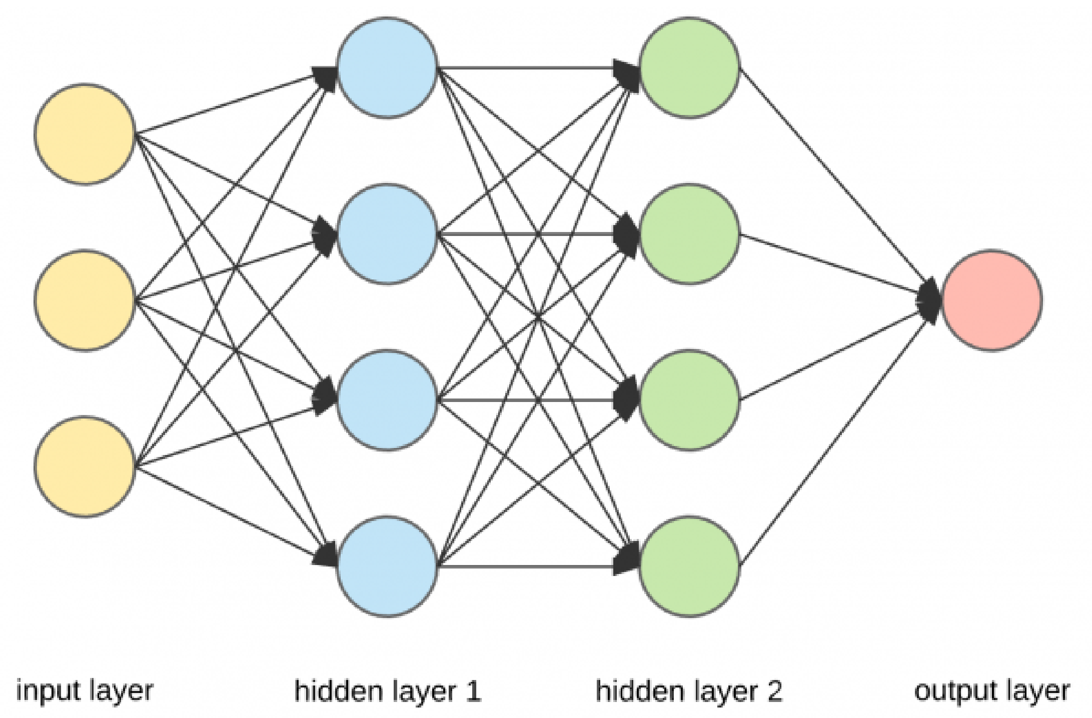


Figure 1.3 High-level illustration of a neural network architecture.
Source: Lelli [17]

Nowadays, neural networks are one of the most common artifacts produced by machine learning algorithms. The large number of layers in the network enable them to learn more complex patterns. Since the natural language abilities are quite complex and consist of many different patterns at the same time, deep neural networks are a fitting way to learn those patterns. Language models represent example usage of these deep neural networks and their training procedure is quite similar to the discussed above.

Let us now dig a bit deeper into the different architectures of language models. As discussed before, Transformer was a pivotal architecture in the natural language processing field that allowed better modelling of natural language. The Transformer architecture consists of the encoder, which encodes textual input, and the decoder, which decodes encoded information and textual input to the output sequence.

The encoder layer consists of the feed-forward sub-layer as well as the Multi-Head Self-Attention mechanism, which analyzes the input from multiple perspectives through its attention heads.

The decoder layer consists of the feed-forward sub-layer, masked multi-head attention that helps to attend to the input autoregressively (one token at a time), and the multi-head attention that attends to the output from the encoder block (contextual information from the whole input sequence). The detailed vanilla Transformer architecture can be seen in figure 1.4.

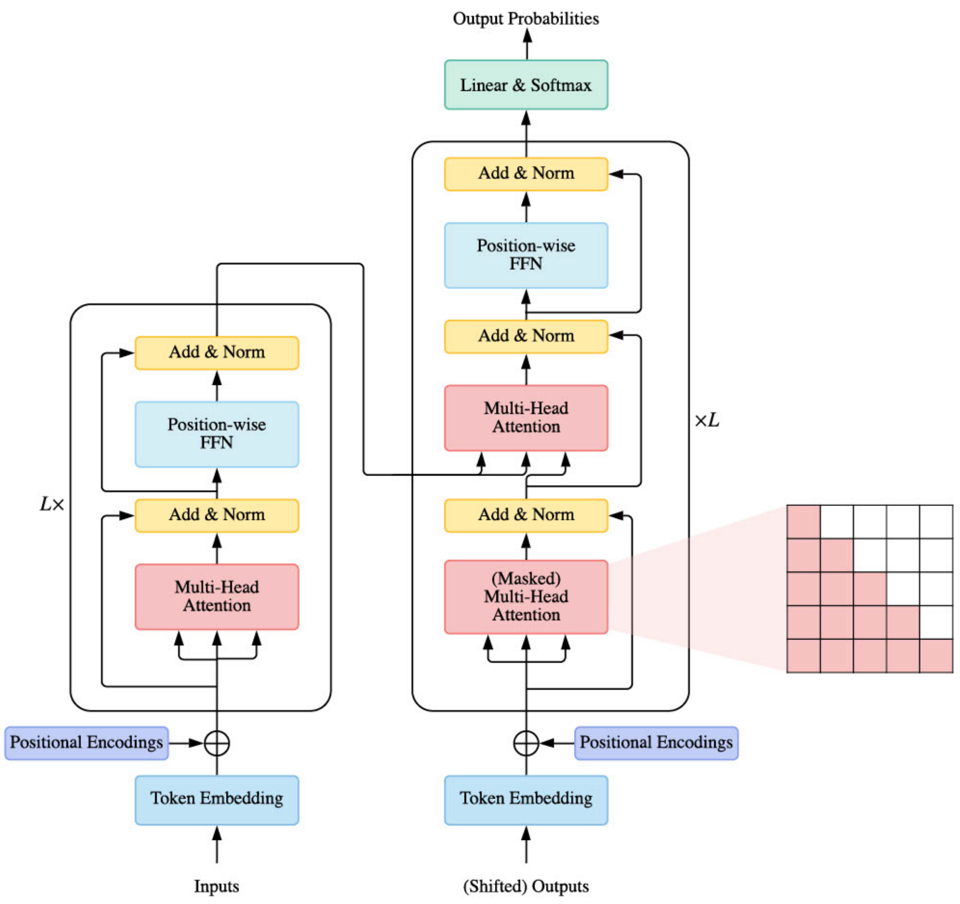


Figure 1.4 Vanilla Transformer architecture.
Source: Lin et al. [18]

The Transformer architecture gave an idea for the subsequent popular language model architectures. Many of these models are built upon each other's ideas. Let us discuss the most notable architecture of a Transformer type: BERT and GPT.

BERT stands for Bidirectional Encoder Representations from Transformers [19]. BERT is a transformer-based encoder-only architecture that is used for various natural language tasks like text classification, named entity recognition, representation learning, etc.

GPT stands for the Generative Pretrained Transformer [20]. GPT also follows the transformer architecture but uses only the decoder. This model is also widely used for a wide variety of the natural language tasks, but all of them have a more text-generative nature.

Transformer-based models are powerful enough to learn complex linguistic patterns from the extensive amounts of textual data. However, we don't need to train the model for the same linguistic abilities over and over again since these trained models were already trained and are available for further training. This initial training on a large amount of data to learn language patterns is called pre-training, and the further training to tweak a model for a specific task, domain or language is called fine-tuning.

The main difference between BERT and GPT is in their architectures, that require a different way of training. BERT needs specific preparation of the data (Masked Language Modelling and Next Sentence Prediction setups [21]). On the other hand, GPT can use raw data (also called unlabeled) and is trained using an auto-regressive fashion. This explains why GPTs have become widely used for large-scale language modelling. Vast amounts of textual data are available online, requiring no annotation and limited preparation. This training setup in machine learning is called unsupervised learning.

This gave rise to Large Language Models. Some researchers believed in the idea of increasing language model capabilities by increasing the size of training data [22]. Figure 1.5 shows how the number of parameters of transformer-based models, mainly GPT architecture, grew over the past years. GPT has provided the ability to use a lot of textual data and train large models with substantial language knowledge. This gave LLMs many strong capabilities, which enabled many powerful applications. Figure 1.5 shows how the number of parameters of transformer-based models, mainly GPT architecture, grew over the past years.

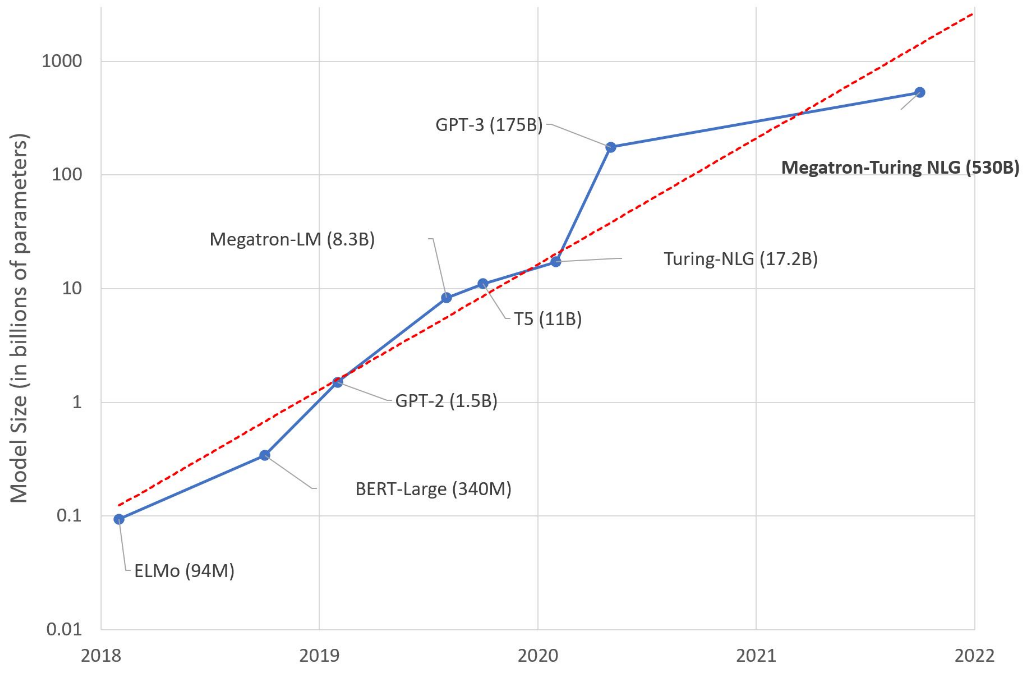


Figure 1.5 Growth of a number of parameters for LLMs over the past years. The Red dashed line indicates a mean trend for shown models.
Source: Simon [23]

1.2.2 Generative Pretrained Transformer

In this section, we will discuss the GPT architecture more in-depth since it is the one that will be used for the thesis experiment

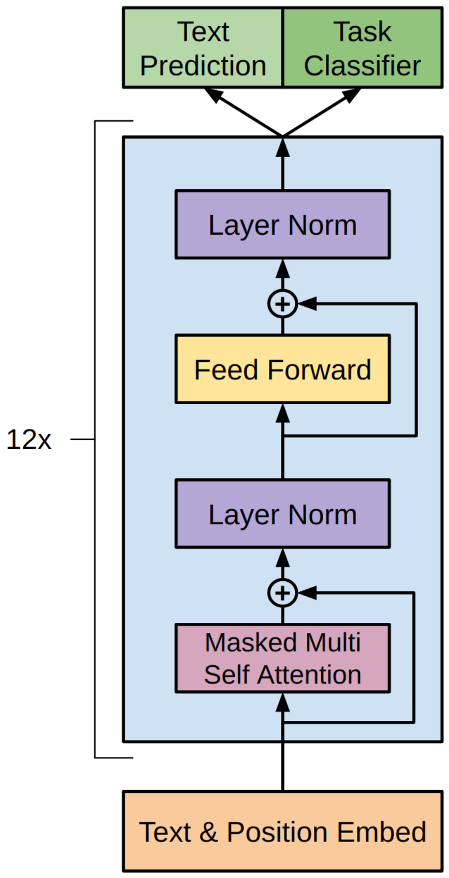


Figure 1.6 GPT architecture.
Source: Radford and Narasimhan [20]

The figure 1.6 shows the architecture of GPT, which represents the decoder block from the vanilla transformer architecture. The GPT algorithm consists of many different components. Before being input, the text is tokenised, which is a process of segregation into smaller segments(tokens) based on rules that consider spaces, punctuation, symbols and words.

The first step is Text Embedding, which is the process when each token is mapped to an embedding vector that captures the semantic meaning of it. This can be achieved by using an embedding matrix when each row corresponds to a token. There is also a Position Embedding, which generates a vector consisting of information about the absolute (in the case of GPT) positions of tokens. It ensures that each token in the input has a fixed and distinct representation that corresponds to its specific position. This is also referred to as the Embedding Layer.

Then, the input representation goes to the decoder block. First, it reaches Masked Multi-Headed Self-Attention. Self Attention is the process of assigning an attention score to each token in the sequence, which is a numerical representation of how the token is important to every other token in an input sequence. Figure 1.7 depicts how the Self-Attention mechanism works. Self-Multi-Headed is an ability to perform Self-Attention computations in parallel. Masking prevents the model from accessing the information from the subsequent tokens, taking only previous ones into consideration during Self-attention. Masked Multi-Headed Self-Attention helps the model predict tokens in an effective and safer manner [18].

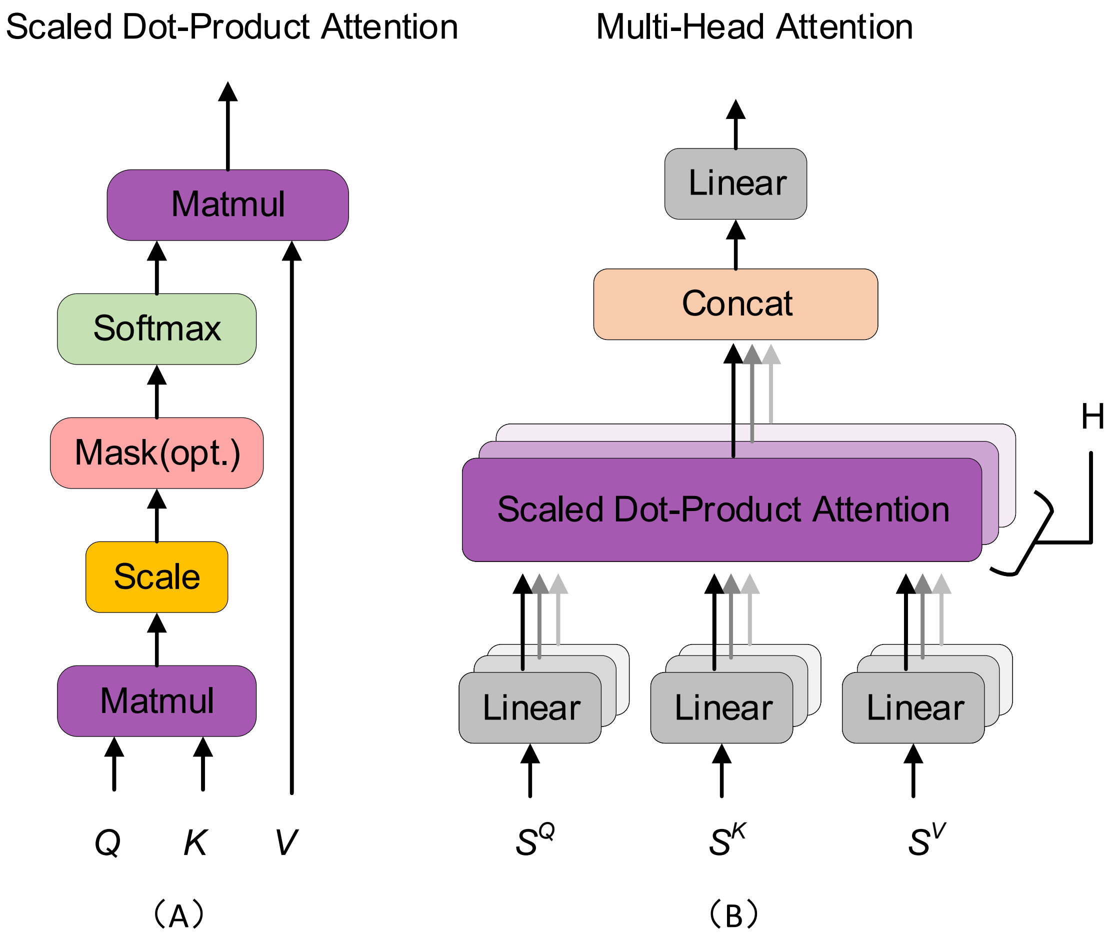


Figure 1.7 Self-Attention mechanism in Transformer. Q here is the query matrix (from the output of the previous decoder layer), K is the key matrix, and V is the value matrix (both come from the output of the encoder). S^Q, S^K,andS^V represent the corresponding source sequence for Q, K, and V.
Source: Zhao et al. [24]

After that, obtained representations pass through the layer where the features are normalised. Then, representations pass through the Feed-Forward neural network layer, and the output is normalised again. This concludes one decoder layer.

Representations pass through 12 decoder layers undergoing the same transformations as discussed above. The resulting output representation will then get translated into a specific token from the available GPT vocabulary. The algorithm predicts the most probable next token for the given preceding sequence of tokens.

Similarly to the process described in the previous section, during the training, the loss function is computed. For most of LLMs including GPT this loss function is represented by the cross-entropy function, which quantifies the difference between the true probability distribution and the predicted probability distribution. Let us look at a Shannon 1.8 entropy formula in more detail:

$$H = -\sum_{i=1}^{C} p(i) \log_2 p(i)$$

Figure 1.8 Shannon cross-entropy formula

Here, H is an entropy, p_i is the probability of the i element, and C is the number of elements. The task of the model during the learning process is to minimise the cross-entropy by adjusting the model's weights and biases. We can interpret the training objective of GPT as follows: it uses training data as a reference in order to predict the most probable continuation of the given text, imitating the reference data and decreasing the difference between training data and generated outputs.

The strength and limitation of GPT is that it was designed to handle text generation tasks in an auto-completion fashion. Many different tasks can be framed as text-generation tasks, but not all tasks are, by their nature, text-generation tasks. For example, a classification task where the primary task is to understand the content of classes and be able to predict a suitable class when rephrased as a text generation task might perform poorer than their encoder-only counterpart, like BERT, which is more suitable for classification tasks due to its training objective. The primary task of GPT is to generate coherent text and less about understanding and creating a proper representation of a text. As a result, it affects the efficiency and quality of such tasks.

1.2.3 OpenAI GPT

GPT is the architecture introduced by the Open AI company research team [25]. OpenAI was one of the pioneer companies in training very large language models that are accessible through an API. There are a lot of different LLMs similar to Open AI GPT introduced by other companies such as Mistral AI [26], Gemini by Google AI team [27], Cohere [28], Llama by Meta AI [29], etc.

Open AI GPT was chosen for the experiment because it is frequently updated, accessible and well-researched. It also has a convenient and customizable API with an option to choose an output format. For the thesis, we will mostly use the most accessible GPT-3.5 turbo model and the new at the time of writing GPT-4o [11]. The queries to GPT will be made through the Open AI Chat Completion API [16].

OpenAI Chat Completion GPT API has many different input parameters that affect the interaction process as well as the output. We will discuss some of them which are important for the thesis experiment:

- **temperature.** This parameter is responsible for query response diversity. The higher the temperature, the more likely the model is to choose a less probable next token. Setting the temperature high makes GPT produce unusual and creative results with a higher probability. However, it also may lead to a broken structure or hallucinations (nonsensical or nonexistent output) [30].
- **response_format.** This is a relatively new OpenAI API feature that allows users to choose an output format. Currently, only 2 options are available: text and JSON. Choosing JSON format guarantees output to be in a JSON format [16].
- **model.** Parameter that sets the exact GPT model that will be used during the model work. For this thesis, we will use the models *gpt-3.5-turbo-0125* and the newest at the time of writing *gpt-4o*. They differ in speed, relevance and size of the training dataset, price, etc. These models might also differ in architecture, but they are closed source, and we cannot make a reliable conclusion about that [16].

1.3 PCG

PCG is a broad concept we discussed more thoroughly in the Introduction. PCG for video games has a very extensive range of applications, supported by various well-researched techniques, both newly created and adapted from other fields. For instance, the original Wave Function Collapse algorithm [31] used for tilemap generation, or L-systems, initially implemented to describe the behaviour of plant cells to model the growth process, were adapted to be used in game development for dynamic level generation in virtual reality [32] or for scenario generation [33].

However, in this section, we will focus on exploring various approaches for generating quests, characters, and stories—elements that are usually represented by natural language. We will analyse these approaches considering that we require the generated textual content for our game to have a strict structure. We will also showcase some great examples of games that significantly rely on PCG techniques to generate entire Game Worlds and narratives. Additionally, we will explore some approaches that use LLMs for PCG.

1.3.1 Narrative World Generation

The concept of world-building with PCG is not new. It involves creating an entire Game World or its significant parts, such as locations, scenarios, or other fundamental game systems that resemble the Game World at some scale. An example of such an approach could be generating a Game World in Minecraft (2011) that includes terrains, biomes and enemies, which resembles a player-driven world [34]. However, there are approaches more relevant to this thesis that attempt to build feature-complete story-driven Game Worlds.

One of the approaches worth discussing is the Narrative World generation algorithm introduced by Balint, J. Timothy and Bidarra, Rafael [35]. This algorithm represented in figure 1.9 takes as input a cohesive and structured story and a knowledge base containing data about objects described in this story. The story is an ordered sequence of actions between objects in given interrelated locations that are assumed to be fully supported by the given knowledge base. The knowledge base not only contains all the objects, locations, and possible relations between them mentioned in the story but also provides objects that are essential for the action in the given location, like a bed for a character sleeping at their house.

In addition, the knowledge base contains data about Motifs, which are the set of objects and relations between them that are typical in the given location but not directly interacted with in the story. Motifs are probabilistic in nature and contain weights for certain objects or relations to occur. Additionally, these weights are affected by the objects that are necessary for the story. This results in a diverse set of generated narrative worlds for the given story. For example, a bedroom motif can contain beds, lamps, chairs, nightstands, or rugs. The bed, in this case, will have the highest probability of occurring, while the chair most likely won't be in every bedroom.

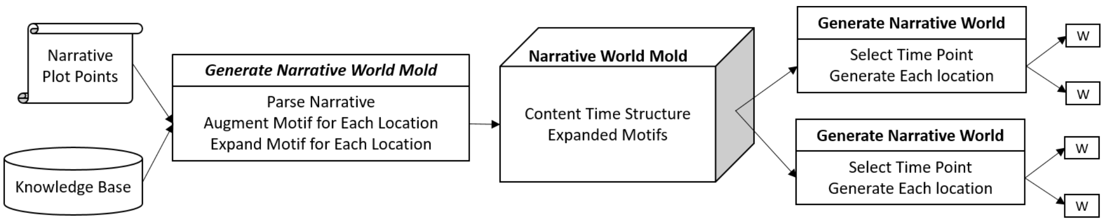


Figure 1.9 Narrative world generation pipeline.
Source: Balint and Bidarra [35]

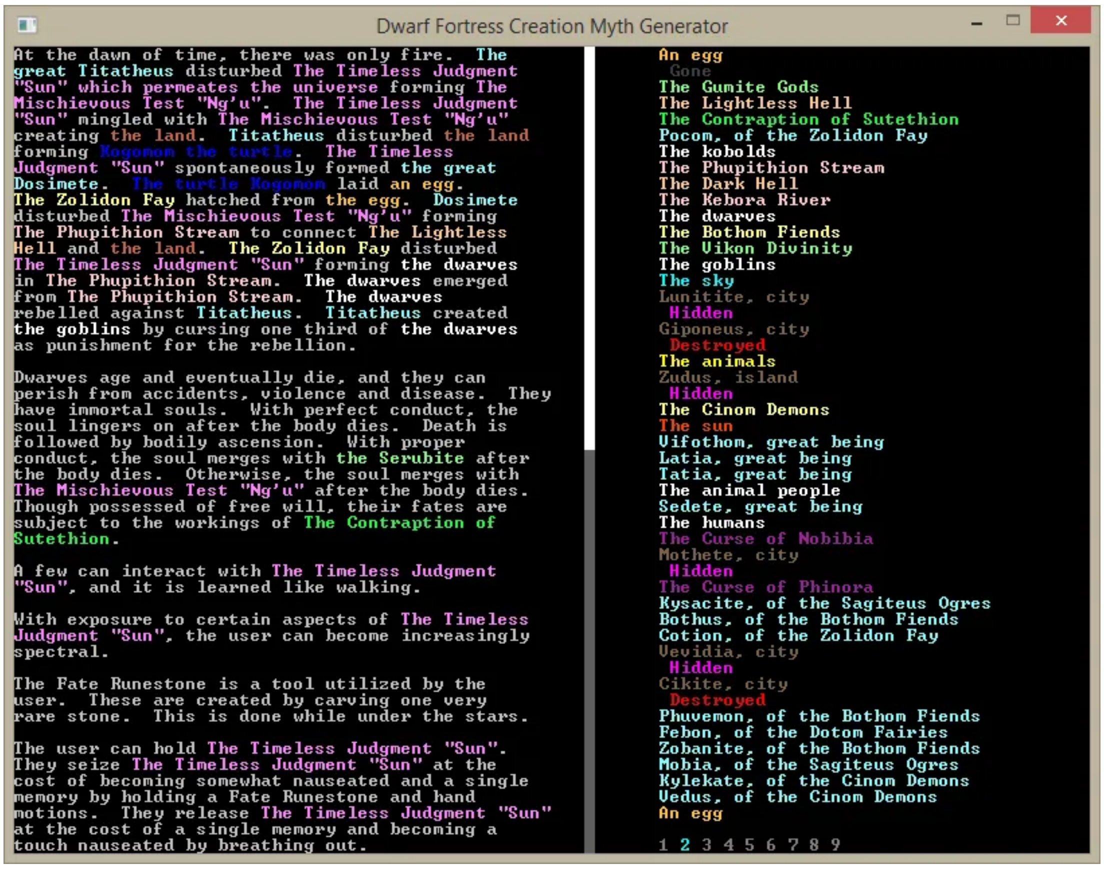
The resulting narrative world for each slice of time contains the state of the locations, characters, and objects that belong to them for the respective points in the story timeline. However, these locations should already have content that may appear in future parts of the story, like a table on top of which the candle can be placed.

This paper gives great insight into what is important for building a world that is dependent on the story.

1.3.2 Dwarf Fortress

An iconic example of utilizing PCG to generate a Game World is Dwarf Fortress (2006). PCG is used to generate most of the elements in the game, including the story, characters, terrain, events, etc. The story that is being generated can be interpreted as a description of the simulation of thousands of sophisticated AI agents or how it was described by the author of the game, Tarn Adams, in the interview *Q&A: Dissecting the development of Dwarf Fortress with creator Tarn Adams* [36] "There's a giant zero-player strategy game going on with somewhat loose turn rules and bad AI (but thousands of agents), and history is just a record of that.".

In the later iterations of the game, the author introduces myth generation, which explains how each key piece of the world (races, zones, primordial entities) appears in this world. The algorithm behind this is similar to one that is responsible for world history but introduces a few more inputs and showcases how the existing procedural generation system for world generation can be expanded for the existing domain [37]. An example of it is shown in figure 1.10. It illustrates how different entities relate to each other by highlighting the colour. Dwarf Fortress introduces many different input values that affect the world. The world generation is seed-based, so each generation can be replicated. This allows players to share interesting worlds or replay them to obtain different outcomes.



```
At the dawn of time, there was only fire.  The
great Titatheus disturbed The Timeless Judgment
"Sun" which permeates the universe forming The
Mischievous Test "Ng'u".  The Timeless Judgment
"Sun" mingled with The Mischievous Test "Ng'u"
creating the land.  Titatheus disturbed the land
forming Kogomom the turtle.  The Timeless
Judgment "Sun" spontaneously formed the great
Dosimete.  The turtle Kogomom laid an egg.
The Zolidon Fay hatched from the egg.  Dosimete
disturbed The Mischievous Test "Ng'u" forming
The Phupithion Stream to connect The Lightless
Hell and the land.  The Zolidon Fay disturbed
The Timeless Judgment "Sun" forming the dwarves
in The Phupithion Stream.  The dwarves emerged
from The Phupithion Stream.  The dwarves
rebelled against Titatheus.  Titatheus created
the goblins by cursing one third of the dwarves
as punishment for the rebellion.

Dwarves age and eventually die, and they can
perish from accidents, violence and disease.  They
have immortal souls.  With perfect conduct, the
soul lingers on after the body dies.  Death is
followed by bodily ascension.  With proper
conduct, the soul merges with the Serubite after
the body dies.  Otherwise, the soul merges with
The Mischievous Test "Ng'u" after the body dies.
Though possessed of free will, their fates are
subject to the workings of The Contraption of
Sutethion.

A few can interact with The Timeless Judgment
"Sun", and it is learned like walking.

With exposure to certain aspects of The Timeless
Judgment "Sun", the user can become increasingly
spectral.

The Fate Runestone is a tool utilized by the
user.  These are created by carving one very
rare stone.  This is done while under the stars.

The user can hold The Timeless Judgment "Sun".
They seize The Timeless Judgment "Sun" at the
cost of becoming somewhat nauseated and a single
memory by holding a Fate Runestone and hand
motions.  They release The Timeless Judgment "Sun"
at the cost of a single memory and becoming a
touch nauseated by breathing out.
```

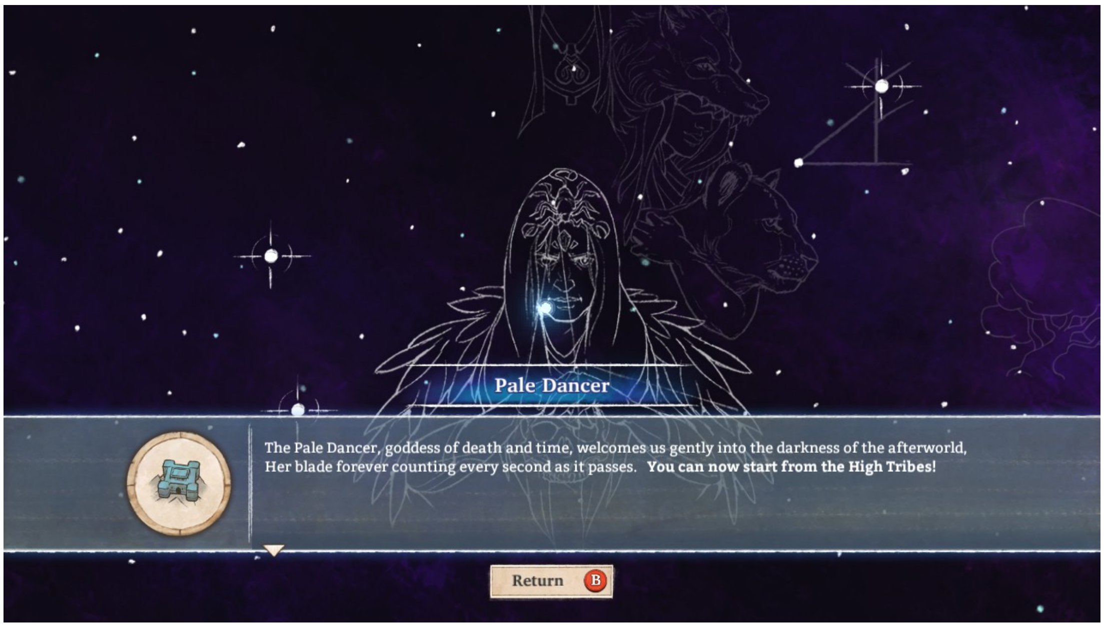
```
An egg
 Gone
The Gumite Gods
The Lightless Hell
The Contraption of Sutethion
Pocom, of the Zolidon Fay
The kobolds
The Phupithion Stream
The Dark Hell
The Kebora River
The dwarves
The Bothom Fiends
The Vikon Divinity
The goblins
The sky
Lunitite, city
 Hidden
Giponeus, city
 Destroyed
The animals
Zudus, island
 Hidden
The Cinom Demons
The sun
Vifothom, great being
Latia, great being
Tatia, great being
The animal people
Sedete, great being
The humans
The Curse of Nobibia
Mothete, city
 Hidden
The Curse of Phinora
Kysacite, of the Sagiteus Ogres
Bothus, of the Bothom Fiends
Cotion, of the Zolidon Fay
Vevidia, city
 Hidden
Cikite, city
 Destroyed
Phuvemon, of the Bothom Fiends
Febon, of the Dotom Fairies
Zobanite, of the Bothom Fiends
Mobia, of the Sagiteus Ogres
Kylekate, of the Cinom Demons
Vedus, of the Cinom Demons
An egg

1 2 3 4 5 6 7 8 9
```

Figure 1.10 Dwarf Fortress myth generator.
Source: *Dwarf Fortress' creator on how he's 42 % towards simulating existence* [38]

1.3.3 Moon Hunters

Another great example of how the narrative generation can be integrated into the game to enhance player experience and engagement is the Myth Generation system in the rogue-like game Moon Hunters. In this talk [37], Tanya X. introduces the algorithm of creating a myth about the played character based on the traits that were acquired during a playthrough. These traits are based on the player's deeds, the conditions of those deeds and relations with in-game characters.



The Pale Dancer, goddess of death and time, welcomes us gently into the darkness of the afterworld, Her blade forever counting every second as it passes. **You can now start from the High Tribes!**

Figure 1.11 Moon Hunter constellation panel.
Source: Short and Adams [37]

However, generating a chunk of the story or a description based on a complex system should result in something more than just a plain text. For that, the game introduced a constellation panel containing all the myths (which describe the player's deeds) the player produced during their previous walkthroughs. Every myth affects the world's generation, and this can be tracked on the constellation map. This makes players learn more about this system and affects their perception of the game during each playthrough. It also adds an additional completionist target to find each star and constellation that is responsible for different player deeds and interactions.

1.3.4 Personality design

The character creation process is a nontrivial task and usually consists of several important steps, such as visual design, assigning personality traits, developing a backstory, etc. We will focus on personality design and generating a backstory that helps integrate the character into the game narrative.

In the book by Tanya X. Short and Tarn Adams [39], the authors explore the topic of character design. In their chapter about Personality Generation, they brought up an important distinction between two types of how generated character traits can be represented in the game. One is the "Character Judgement" (Behaviour First) method, in which players draw conclusions about their personality based on the character's words or actions. Second is the "Tell Then Show" (Reasoning First) approach, in which players are informed about significant personality traits. That information might be usually provided for players to then align their decisions made in interactions with these characters. Schemes for these character representation patterns are shown in figure 1.12.

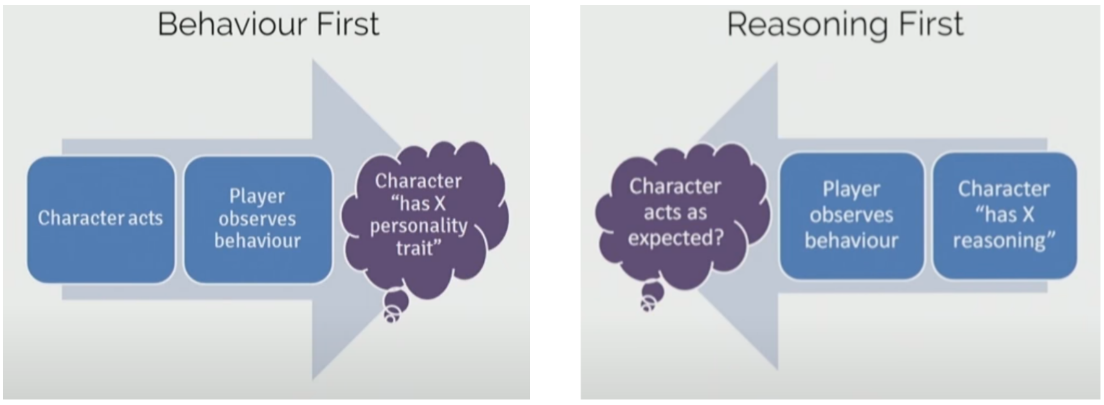

Behaviour First
Character acts
Player observes behaviour
Character "has X personality trait"
Reasoning First
Character acts as expected?
Player observes behaviour
Character "has X reasoning"

Figure 1.12 Two approaches on how character personality can be represented in a game.
Source: Short [40]

The Reasoning First approach is used in games with procedurally generated characters, like Dwarf Fortress 1.3.2 or Crusaders Kings 2. This thesis will use the Behaviour First approach, which is more classic for hand-created characters. The language model interprets traits assigned to characters to generate a dialogue tree that is then assessed by players.

1.3.5 Practical PCG Through Large Language Models

We explored some PCG techniques that can help developers craft a Game World. We also discussed a few techniques and examples of how the generated narrative can be inserted into the game. Here, I would like to give an example of how LLMs can be used to generate game parts that are not directly associated with natural language.

In their paper, Muhammad U Nasir and Julian Togelius [41] showed an interesting approach to generate levels for their top-down rogue-lite game Metavoidal using the pre-trained language model GPT-3. In their algorithm 1.13, they represented the data as a sequence of tokens described in the prompt and an instruction that there should be a certain percentage of walkable tiles. Here is how the string representing a 3x3 tile fragment can look in a prompt. Here A represents a walkable tile, # is a wall, J is a door, $\backslash n$ represents a new line, and % is an end of a sequence 1.1.

$$"AAA\backslash n\#J\#\backslash nAAA\backslash n\%" \tag{1.1}$$

The levels have some constraints regarding the positions of the walkable and unwalkable tiles and doors. The prompt itself does not contain any instruction on that matter, and this constraint was only overcome by continuous training with valid data.

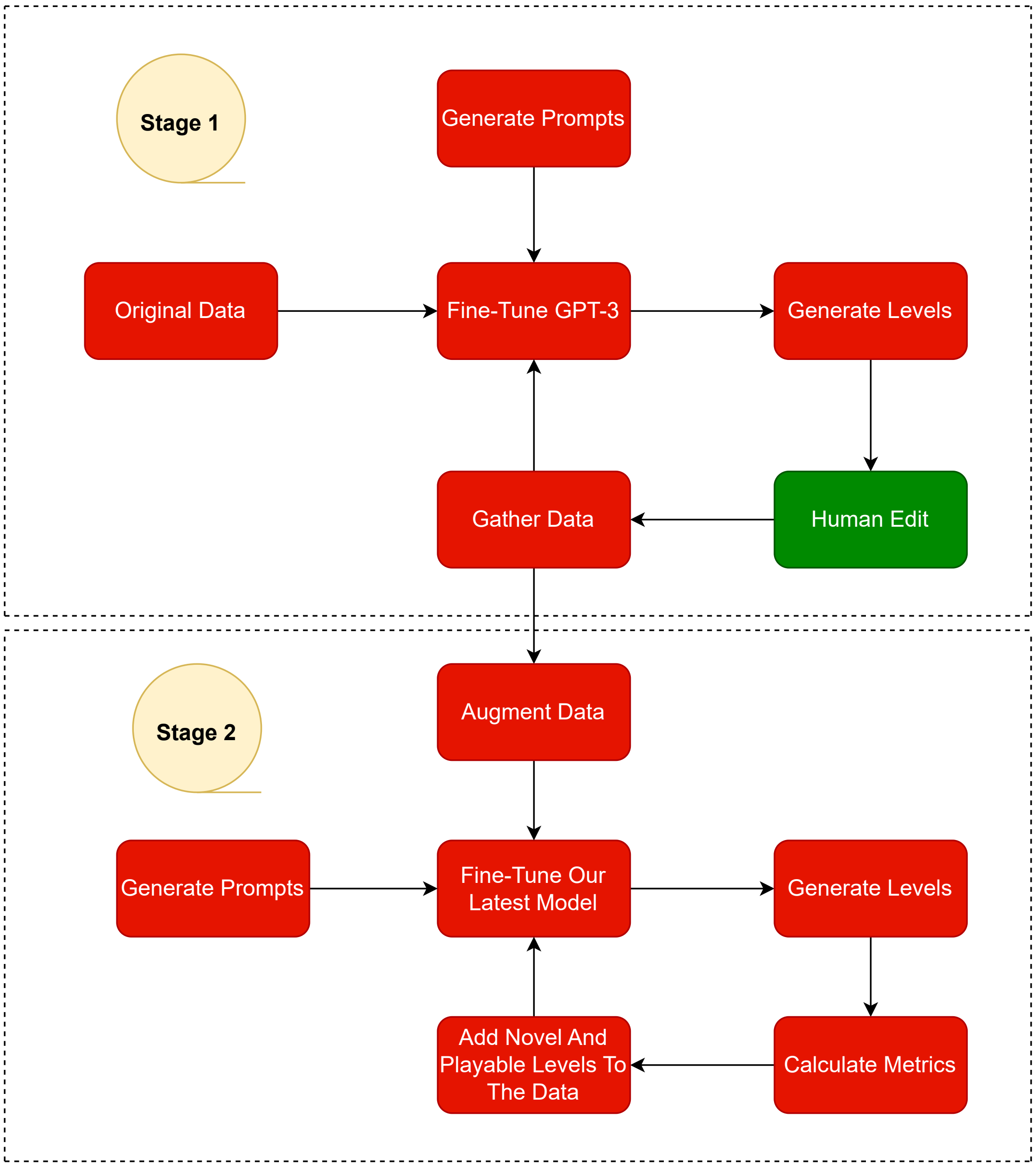


Figure 1.13 A flowchart of how both stages work. The yellow circle indicates the stage number. Red color indicates automated process and green indicates human-in-the-loop process.
Source: Nasir and Togelius [41]

Two different metrics were used to evaluate the result. First, that the level should be playable-novel. To fulfil this, the level should be playable, which is true if all constraints are passed. It should be novel, for which the levels in a data set are compared to each other and should have a calculated difference of less than the chosen threshold. In this experiment, it's 10% of the string length. The second metric is accuracy. Which is how accurately the level followed the instructions provided in a prompt. This work gives us some notion of how the game data can be interpreted to insert it into a prompt for a language model. However, for our thesis, we will consider a different approach.

2 GPT JRPG

During the initial experiments with GPT, several features of the game suitable for the thesis experiment were considered:

- The game genre should fit the idea of generating distinct Game Worlds. That means that the game should be abstract enough. What exists in the game should be able to represent a wide variety of fantasy narratives. That possibly could be achieved by creating an enormous number of different game mechanics and assets, which is not achievable considering the scope of the thesis. The preferred way is to make a simple game that abstracts most of the objects, mechanics, or locations that might exist in the textual part of the game generated by GPT.
- One of the main strengths of the Open AI chat completion API is generating dialogues. Therefore, it would be beneficial to have dialogues as a meaningful part of the game.
- One of the initial ideas for content generation with LLMs was to make it choose from the list of created assets based on the already generated part of the game and the description of the hand-created game features. The game should have a significant amount of assets. However, they should be stylistically compatible because the instructions for the model would allow it to use them in any combination it finds meaningful.
- The game should have a sufficient number of mechanics and depth to show that the conclusions drawn from this thesis's results can be scaled and compared to commercial games.

After several iterations, the JRPG game genre was chosen, and its advantages were described in Chapter 1.1. The nominal name "GPT JRPG" was chosen to highlight the game's purpose and genre.

The core gameplay consists of moving along the levels and between them, interacting with enemies, completing quests, and managing units to try forming a powerful and balanced group. The main objective of the game is to defeat the world antagonist. The difficulty of the game and the necessity of adapting the playstyle comes from the fact that after the fight, units are not healed. That requires players to choose battles that are completing the quests which can provide an item that might help to recruit new units or heal the group.

2.1 Designing the Game

In this section, we will go through design decisions made for GPT JRPG and discuss important game mechanics and entities. As the game engine, we have chosen Unity [42] because it is free for the scale of the game we made and has a lot of freely accessible information and also because it uses C# as a main scripting language, which aligns with the prior experience of the author.

Initially, the game was designed around the data received when communicating with GPT, and the first objects that were created are **World Enemy**, **Protagonist**, and **Units**. The next step in designing the JRPG was to create a turn-based Battle System to have a playable Vertical Slice (functional game piece showcasing the intended player experience) of the game.

After that, GPT JRPG was iteratively improved by alternately updating the algorithm of interaction with GPT API and adding new features to the game that can represent newly produced data. Here, we will talk about the most important entities that are using data generated by GPT:

- **Level**. The game has three levels, each characterised by its components. The level contains world enemies, the protagonist, a level door, and quests. Each level has a fixed size and identical landscape shape.
- **Protagonist**. This is a playable character that has descriptive characteristics which can be found in a knowledge base. The player has a friendly unit group under control and can have usable items.
- **World Enemies**. They are the level objects the protagonist can interact with. The interaction starts with a dialogue. Depending on the dialogue outcome, the interaction can result in recruiting, ignoring or battling with the unit group assigned to the corresponding enemy.
- **Level door**. The door to the next level opens when the main quest of this level is completed, it doesn't exist on the last level.
- **Quest system**. There are main quests that must be completed to finish the game and side quests that are optional. Side quests give an item as a reward. There are three types of tasks in quests: "kill", "unite", and "kill or unite". To fulfil the kill task, an indicated enemy should be defeated in the battle. "Unite" can be done by recruiting the enemy through dialogue. "Kill or unite" just indicates that the quest can be completed by both actions. Quest fails if the enemy for the task "unite" is killed or for the task "kill" is recruited. The side quests of the current level also fails when moving to the next one. The quest list can be found through a pause menu.
- **Unit**. It is a generated character with an assigned class, power level, hp, armour and damage. Each unit has a special ability corresponding to its class.
- **Unit Group**. The group consists of several units that can be engaged in the battle. The protagonist group can be found and controlled through a pause menu. The game is lost when zero units remain in the protagonist group. Each enemy unit group is represented by the world enemy. Protagonist units can be swapped during the recruiting process.
- **Dialogue**. Each world enemy engages in the dialogue upon interaction with the protagonist with one of three outcomes. The first is the enemy, forcing

the protagonist to fight. The second is ignoring, meaning the protagonist can retreat in peace or engage in battle. The third is that the enemy is ready to unite, which allows the protagonist to recruit its units, but the protagonist can still engage in battle or ignore.

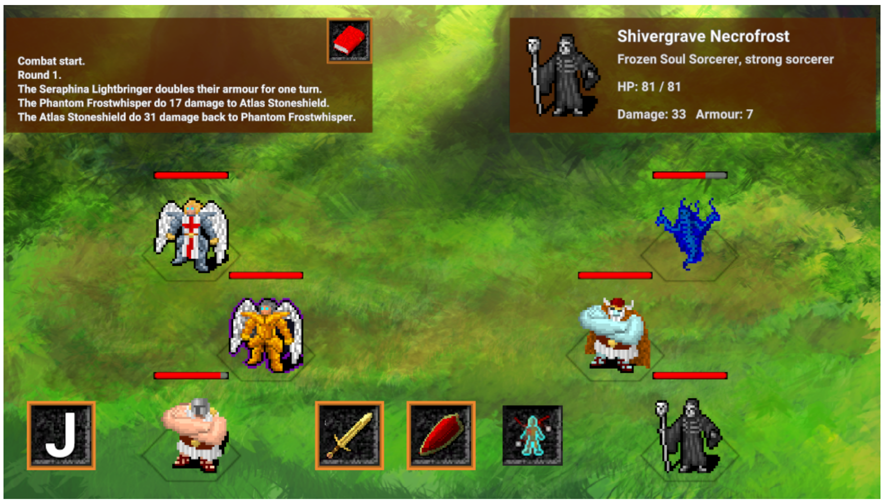
- **Items**. There are two types of items: amulet of healing and amulet of alliance. Both are stackable and can be acquired through side quests. Amulet of healing can be used to fully heal the protagonist group. Amulet of alliance can be used to force world enemies to unite with the protagonist if the current outcome doesn't allow it.

2.2 Battle system

The battle system is one of the core gameplay mechanics in which the protagonist group engages in a fight against the world enemy group. Combat is turn-based and structured into rounds. The round ends when every protagonist unit performs its action. Even though GPT is not used for battle system directly, it is still built using GPT generated units and their characteristics. There are three types of action:

- **Attack**. It deals damage to a chosen enemy unit.
- **Armour up**. It doubles the armour value for this unit for one turn.
- **Class ability**. the ability is special for each class. It can be a powerful spell, a mighty attack or a passive skill, we will discuss them in more detail in the next section.

Every action taken during the combat is documented and written in the battle journal, which can be viewed in the UI. Combat ends when all units from one of the sides are dead. The graphical representation of the battle scene can be seen in the figure 2.1.


Combat start.
Round 1.
The Seraphina Lightbringer doubles their armour for one turn.
The Phantom Frostwhisper do 17 damage to Atlas Stoneshield.
The Atlas Stoneshield do 31 damage back to Phantom Frostwhisper.

Shivergrave Necrofrost
Frozen Soul Sorcerer, strong sorcerer
HP: 81 / 81
Damage: 33 Armour: 7

Figure 2.1 Battle scene in GPT JRPG.
Source: Custom screenshot from GPT JRPG

2.3 Unit classes and power levels

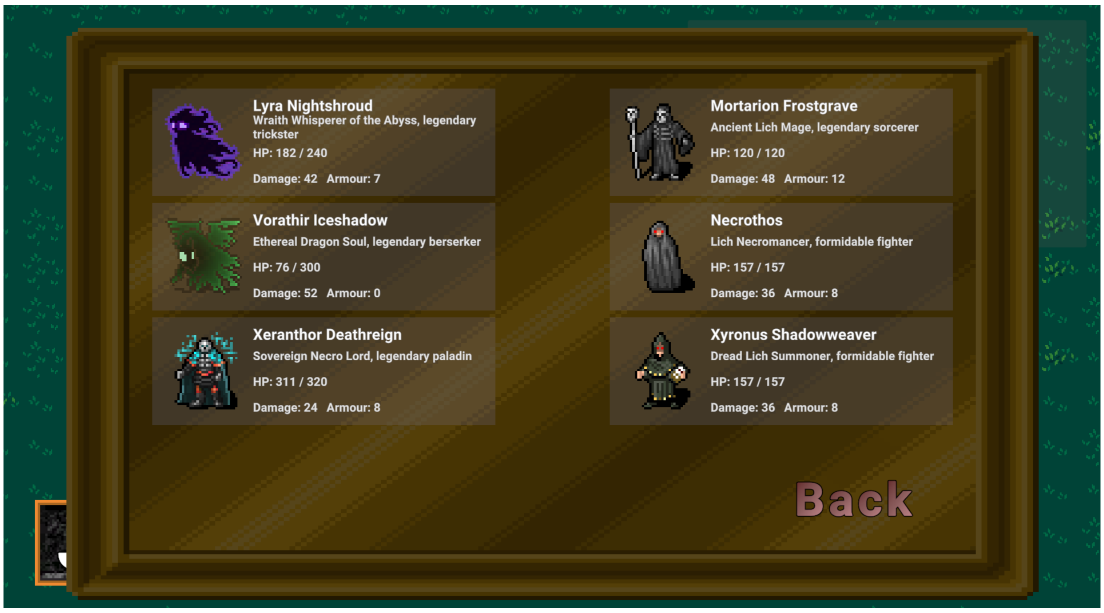
There are 10 classes in the game, each having a special ability that can be used during the combat. On top of that, there are also 7 power levels that have different amounts of health points, damage and armour. Each class also has a different stat distribution. The exact distribution of stats can be found in the Attachment A.1. Here are descriptions of each class:

- **Fighter**. Attacks back when being attacked by the enemy unit. Passive ability. Has balanced characteristics but is slightly focused on damage.
- **Sorcerer**. Stuns enemy unit for 1 turn. Cooldown: 1 turn. Prioritises damage.
- **Paladin**. Heals all ally units for 15% of their max health. Cooldown: 4 turns. Prioritises health.
- **Protector**. Taunts all enemy units for 1 turn. Cooldown: 2 turn. Excels in health, moderate armour, and damage is extremely low.
- **Bastion**. Activates armour up for all ally units for 2 turns. Cooldown: 3 turns. Prioritises Armour.
- **Healer**. Heals ally unit for 20% of their max health. Cooldown: 1 turn. Has balanced characteristics.
- **Trickster**. Makes an enemy unit attack another enemy unit the next turn. Cooldown: 4 turns. Has balanced characteristics, slightly focused on health and damage.
- **Berserker**. Attacks all enemy units, also hurting oneself. Cooldown: 3 turns. Doesn't have any armour. Health and damage are balanced.
- **Marksman**. Attacks enemy units, ignoring their armour. Cooldown: 1 turn. Doesn't have any armour. Damage is highly prioritised.
- **Shaman**. All ally units have 50% to avoid damage for 1 round. Cooldown: 4 turns. Has low armour. Health and damage are balanced.

2.4 UI

UI is an important part of the game. It allows the player to receive information about the current Game World and perform certain actions. Here is how the UI is constructed in the game.

- **Pause Menu:**
  - **Quest List**. Here, players can find all the information about the quests. Each quest has a colour indentation, whether it is completed (green), failed (red) or currently available (white). By clicking on the quest more detailed information can be found.
  - **Group Info**. The current protagonist group data is displayed in the group info menu. The units' positions here correspond to those in battle and can be swapped. Information about the items and units that can be healed with the amulet of healing can also be found here.
- **Knowledge Base:**
  - **World**. The story of the world and description of each level.
  - **Hero**. The description and backstory of the protagonist.
  - **Classes**. Description and cooldown of each unit class ability.
  - **Bestiary**. List of all units existing in the world.
- **Other elements:**
  - **Game message**. The message is shown when the game starts, lost or completed.
  - **Dialogue interface**. Open upon interacting with the world enemy. During the dialogue, the player answers to the world enemy phrases by choosing from the available answers. At some point, the dialogue ends, and buttons with the possible actions appear depending on the dialogue outcome.
  - **Recruit menu**. Opens after the dialogue with the recruiting outcome. Any protagonist group unit can be swapped with the enemy one by dragging them with a mouse. Depicted in the figure 2.2.


Lyra Nightshroud
Wraith Whisperer of the Abyss, legendary trickster
HP: 182 / 240
Damage: 42 Armour: 7

Mortarion Frostgrave
Ancient Lich Mage, legendary sorcerer
HP: 120 / 120
Damage: 48 Armour: 12

Vorathir Iceshadow
Ethereal Dragon Soul, legendary berserker
HP: 76 / 300
Damage: 52 Armour: 0

Necrothos
Lich Necromancer, formidable fighter
HP: 157 / 157
Damage: 36 Armour: 8

Xeranthor Deathreign
Sovereign Necro Lord, legendary paladin
HP: 311 / 320
Damage: 24 Armour: 8

Xyronus Shadowweaver
Dread Lich Summoner, formidable fighter
HP: 157 / 157
Damage: 36 Armour: 8

Figure 2.2 Recruit Menu in GPT JRPG.
Source: Custom screenshot from GPT JRPG

2.5 Assets

While there are many various assets that are used in the game, here we will only discuss the ones that will be chosen from GPT.

- **Unit assets**. There are 354 unit assets named according to their appearance. These assets are used for both world enemies and units.
- **Terrain tilemaps**. The tiles used to form the world's terrain can have 4 different types of obstacle terrain and 14 types of walkable terrain. So, there are a total of 56 tilemaps that can form a distinct terrain for the level. An example of a tilemap with lava rock as a walkable tile and lava as an obstacle tile can be seen in figure 2.3.

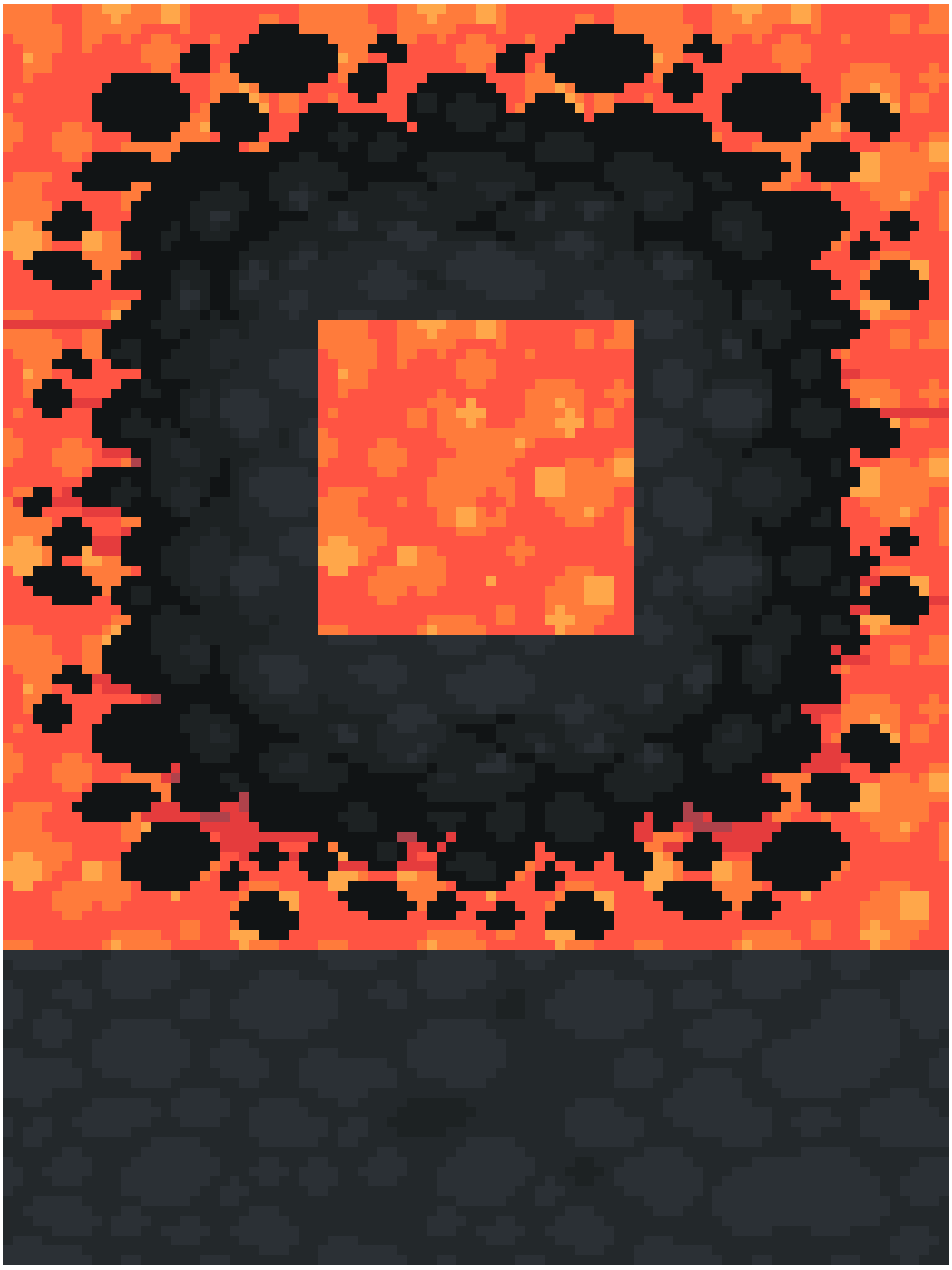


Figure 2.3 Terrain Tilemap from GPT JRPG.
Source: Asset used in GPT JRPG

2.6 World representation

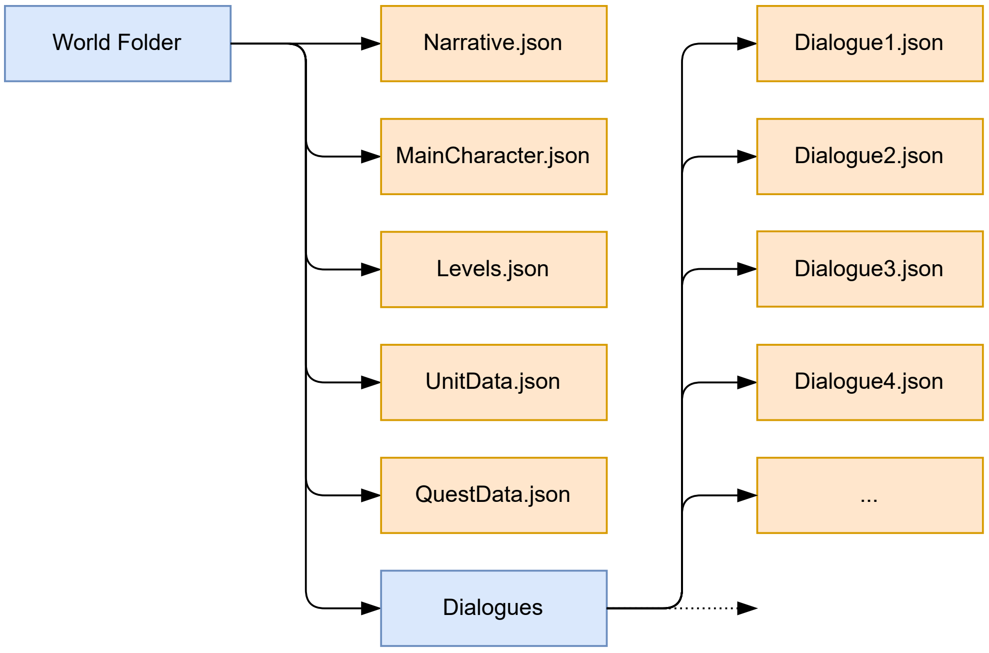


Figure 2.4 Folder structure for a Game World.

In order to provide the generated data to the game as input, it should be in a readable format. While even plain text can be parsed, structured formats such as XML or JSON would be a lot more convenient and error-proof as the game input. Since the only structured output available from OpenAI GPT API is JSON, it was chosen as the preferred response format. How the parameters are assigned to a model is described in this section 1.2.3.

The Game World is generated before the game starts (whether from a main menu or a script that launches an API query to LLM, which will be discussed in the following chapter). This Game World is stored in a folder with the structure shown in Figure 2.4. When the game is launched, all existing world folders are parsed, and buttons for each world are created. When the world is selected, and a player presses the button Start Game, each JSON file in the corresponding folder is parsed, and a C# data structure containing all the generated data is created. This Game World data structure can then be accessed from any game object and is destroyed when players return to a main menu, shown in figure 2.5.

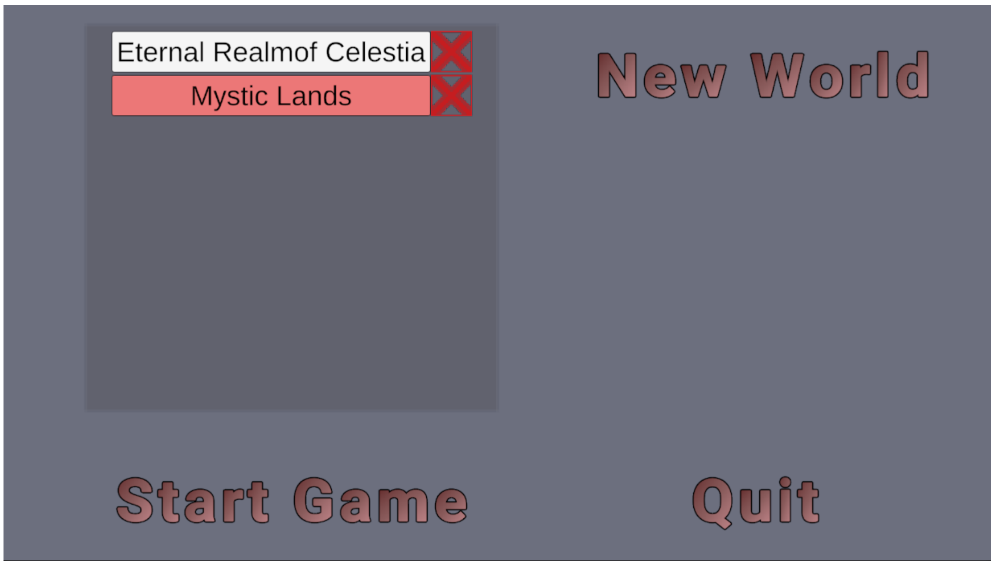


Figure 2.5 Main menu in GPT JRPG.
Source: Custom screenshot from GPT JRPG

3 Interaction with LLMs

In this chapter, we will provide details on the interaction process with the OpenAI GPT API and discuss a few tested approaches. We will start with an overview of the existing theory behind prompting. After that, we will go into specifics about how the interaction with the model is performed for the thesis experiment and then provide our observations regarding the prompt engineering.

3.1 Prompt Engineering

LLMs are a specific case of language models, which suggests that the main way to interact with them is through natural language. Language models that operate on texts try to understand the input text and provide the most probable continuation of the given text. Giving the right input context helps prime the model in the right direction, giving it a direction to a more desirable output.

This specific input to the model got the name prompt following a long-lasting tradition in Computer Science and natural language processing fields. It became widespread after the popularisation of autoregressive LLMs like GPT. The prompt is a query to the LLM that acts as an instruction to give more context to the model for a desirable outcome.

Discussing prompting is not possible without understanding the concept of in-context learning. In this section 1.2.1, we mentioned ways a large language model can be trained using two main techniques: pre-training and fine-tuning. Pre-training is a process of training LLM on a large amount of data so that it can learn its main linguistic abilities and understanding of the world. Fine-tuning, on the other hand, can be used to adjust a model to a specific task or knowledge area and makes a model more suitable to solve very narrow and specialised tasks.

However, fine-tuning might not be necessary for LLMs trained on large datasets of relatively unbiased data because these models have strong in-context learning capabilities [43]. In-context learning is the ability of an LLM to perform specific tasks based only on the information provided in a prompt without modifying the model parameters through fine-tuning. This means that when trained on a large amount of data, the LLM can use the prompt as context, improving the knowledge gained during pre-training to handle unfamiliar tasks by using related abilities developed during training.

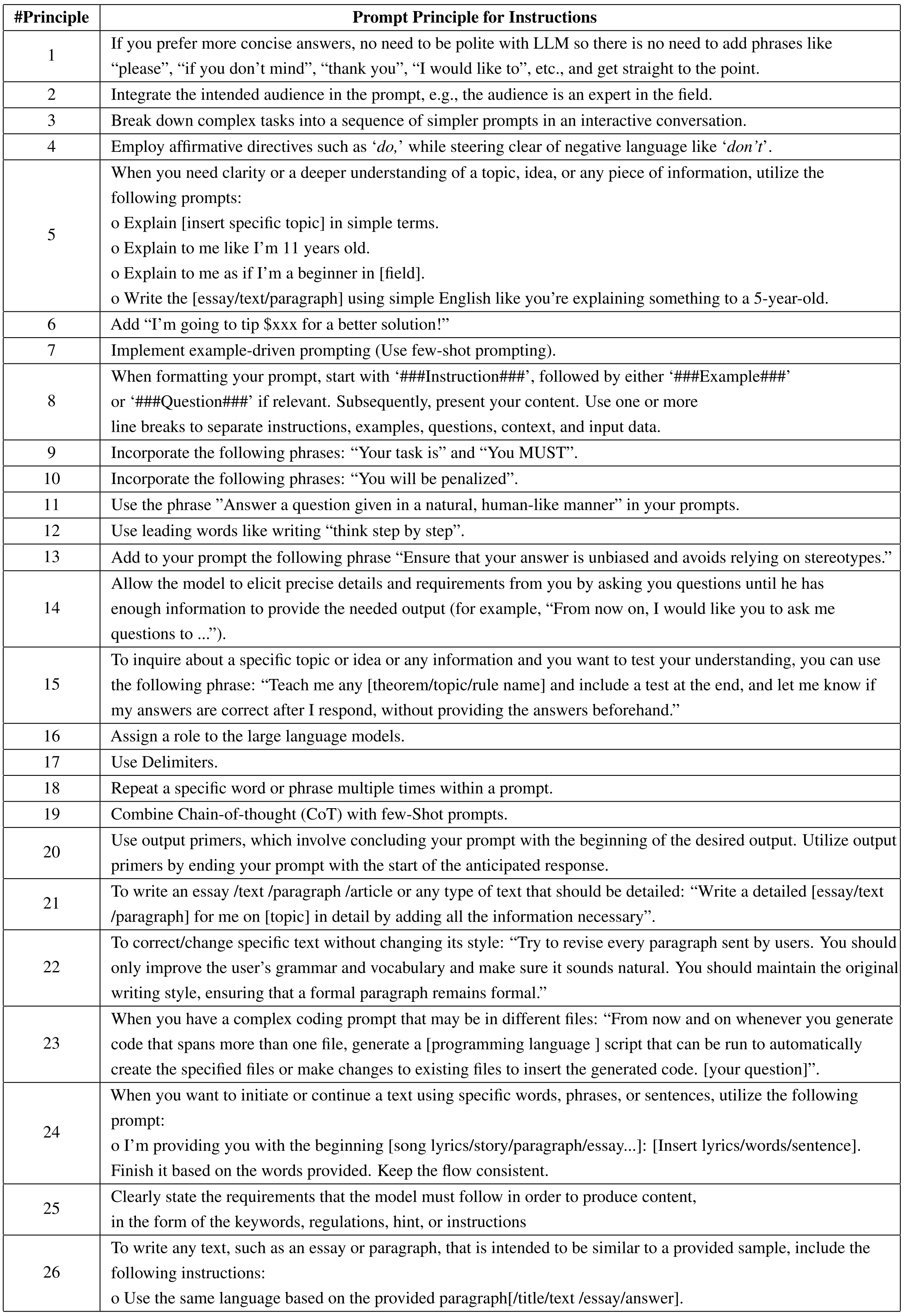
It is worth noting that the way the prompt is phrased largely affects the model's ability to give answers. Even minor changes to the prompt can give drastically different answers. The main reason for that is the data a model is trained on and how it was pre-trained. The data acts as a database, and if one prompt linguistically is more aligned with the database than another one, it has a higher chance of returning the desired answer. The result given by LLM is also highly affected by the clarity of instructions. However, it would be fair to mention that the inner mechanism of LLMs is considered uninterpretable by humans because these models are very large and complex. So, many existing techniques aimed at improving prompt quality are empirical.

Due to the fact that the quality of the prompt highly affects the quality of the LLM response led to a new practice called Prompt Engineering. Prompt engineering is a set of techniques to design a prompt that aims to optimise the AI responses. Prompt engineering is a relatively new term, but it has already got a lot of attention in the research community, and a lot of work has already been done studying and developing effective techniques [44].

There are many techniques which focus on having a specific structure or order, avoiding specific word choices or including specific phrases or formatting, as well as providing in prompt-specific settings like bribing, setting roles or providing audience. There are a lot of different opinions on which techniques are better or how to interpret them. Sometimes, prompt engineering overviews might have conflicting opinions regarding approaches. One of the overviews of these techniques can be found in figure 3.1.

However, some prominent prompt engineering techniques representing a specific prompt engineering concept or paradigm are worth discussing. We also found some of them useful for experimentation. The techniques that we want to discuss are zero-shot learning, few-shot learning, prompt chaining, chain of thought, and retrieval-augmented generation. This list is not exhaustive, but we will provide more details on several prominent concepts to illustrate prompt engineering techniques.

While prompt engineering also refers to interaction with, for example, image or video-generating models, we will focus on those most relevant to language models.



#Principle	Prompt Principle for Instructions
1	If you prefer more concise answers, no need to be polite with LLM so there is no need to add phrases like "please", "if you don't mind", "thank you", "I would like to", etc., and get straight to the point.
2	Integrate the intended audience in the prompt, e.g., the audience is an expert in the field.
3	Break down complex tasks into a sequence of simpler prompts in an interactive conversation.
4	Employ affirmative directives such as '*do,*' while steering clear of negative language like '*don't*'.
5	When you need clarity or a deeper understanding of a topic, idea, or any piece of information, utilize the following prompts: o Explain [insert specific topic] in simple terms. o Explain to me like I'm 11 years old. o Explain to me as if I'm a beginner in [field]. o Write the [essay/text/paragraph] using simple English like you're explaining something to a 5-year-old.
6	Add "I'm going to tip $xxx for a better solution!"
7	Implement example-driven prompting (Use few-shot prompting).
8	When formatting your prompt, start with '###Instruction###', followed by either '###Example###' or '###Question###' if relevant. Subsequently, present your content. Use one or more line breaks to separate instructions, examples, questions, context, and input data.
9	Incorporate the following phrases: "Your task is" and "You MUST".
10	Incorporate the following phrases: "You will be penalized".
11	Use the phrase "Answer a question given in a natural, human-like manner" in your prompts.
12	Use leading words like writing "think step by step".
13	Add to your prompt the following phrase "Ensure that your answer is unbiased and avoids relying on stereotypes."
14	Allow the model to elicit precise details and requirements from you by asking you questions until he has enough information to provide the needed output (for example, "From now on, I would like you to ask me questions to ...").
15	To inquire about a specific topic or idea or any information and you want to test your understanding, you can use the following phrase: "Teach me any [theorem/topic/rule name] and include a test at the end, and let me know if my answers are correct after I respond, without providing the answers beforehand."
16	Assign a role to the large language models.
17	Use Delimiters.
18	Repeat a specific word or phrase multiple times within a prompt.
19	Combine Chain-of-thought (CoT) with few-Shot prompts.
20	Use output primers, which involve concluding your prompt with the beginning of the desired output. Utilize output primers by ending your prompt with the start of the anticipated response.
21	To write an essay /text /paragraph /article or any type of text that should be detailed: "Write a detailed [essay/text /paragraph] for me on [topic] in detail by adding all the information necessary".
22	To correct/change specific text without changing its style: "Try to revise every paragraph sent by users. You should only improve the user's grammar and vocabulary and make sure it sounds natural. You should maintain the original writing style, ensuring that a formal paragraph remains formal."
23	When you have a complex coding prompt that may be in different files: "From now and on whenever you generate code that spans more than one file, generate a [programming language] script that can be run to automatically create the specified files or make changes to existing files to insert the generated code. [your question]".
24	When you want to initiate or continue a text using specific words, phrases, or sentences, utilize the following prompt: o I'm providing you with the beginning [song lyrics/story/paragraph/essay...]: [Insert lyrics/words/sentence]. Finish it based on the words provided. Keep the flow consistent.
25	Clearly state the requirements that the model must follow in order to produce content, in the form of the keywords, regulations, hint, or instructions
26	To write any text, such as an essay or paragraph, that is intended to be similar to a provided sample, include the following instructions: o Use the same language based on the provided paragraph[/title/text /essay/answer].

Figure 3.1 Overview of 26 randomly ordered prompt principles.
Source: Bsharat et al. [45]

3.1.1 Prompt Engineering Techniques

In this subsection, we will observe a few of the most significant techniques and approaches to prompting.

- **Zero-shot learning**. This technique uses the prompt instructions and the internal capabilities of large language models to perform a specific task. This means that we don't need to supply any examples but just provide meaningful instructions describing the task in detail to the language model. This technique relies heavily on information provided by a user and knowledge gained during pre-training about different real-world tasks. One of the main advantages of using this technique is that it does not load the model with the additional context, which makes the Transformer-based LLMs work faster in comparison to using techniques that require providing an extended context [46].

- **Few-shot learning**. In contrast to zero-shot learning, this technique requires a few examples to be provided to the large language model in addition to the main instruction to serve as an example for the desired behaviour or outcome for a specific task. Examples like anchors allow LLMs to understand the task and the desired output better. Sometimes, just one example is enough, and this is called one-shot learning. One of the most typical examples of one-shot learning is guiding for output formatting, for example, making an LLM to output data in a specific format like JSON or CSV [47].

- **Prompt chaining**. This technique is commonly used for complex tasks. Prompt chaining allows to chain a sequence of prompts, one after another, to split a complex task into smaller and more manageable subtasks. This technique simplifies the prompt since LLMs struggle with very detailed prompts and chains one after another so that in each subsequent prompt, the information obtained at the previous prompting step is used. A disadvantage of this approach is that it requires providing context about all the previous results and instructions for every prompt. As a result, it will be more expensive and take longer than providing only one prompt.

- **Chain of thought (CoT)**. COT is a technique that uses a step-by-step explanation of performing a specific task in a prompt as an extended instruction [48]. This technique is commonly used for tasks that require reasoning capabilities. Not just providing the correct answer as an illustration to the model of how to proceed with the task but providing explanations in the prompt on how the task should be tackled step by step gives better results.

- **Retrieval Augmented Generation (RAG)**. RAG is a design system that is used to extend the knowledge base of the model without using techniques that aim at changing the model parameters like fine-tuning [49]. Many generation tasks require some specific knowledge base that models could attribute to while responding to users, for example customer support chatbots or information retrieval systems. The knowledge that LLM gained during the pre-training might not be enough to perform a specific task. RAG helps to use an external knowledge base that wasn't incorporated during the pre-training or even fine-tuning procedure. RAG technique suggests adding relevant documents directly to the prompt surrounded by some helpful instructions on how to use these documents, thus providing references for the model to look through.

This allows models to serve more fine-grained information, which is more helpful for users or serves the most recent and up-to-date information. Since the number of documents in the knowledge base can be large, RAG can also define the way how the most useful references should be selected and stored. Most common RAG applications use vector databases that do similarity searches to select candidates. While RAG is able to solve quite specific tasks, this system is very demanding and requires a fast database, efficient retrieval mechanism and query-building algorithm.

For the diploma thesis, we don't need to implement such a complex design system. However, we will still use the prompt engineering technique, which references the knowledge base and includes specific instructions on how the data from this knowledge base is used.

Several of the prompt engineering techniques can be combined to achieve even better performance for LLMs.

3.2 Query structure

During the first iterations, the whole world was generated in one query to OpenAI API. As the world emerged, the query was split into several consecutive ones. This helped reduce the total request processing time and to receive more reliable and correct answers.

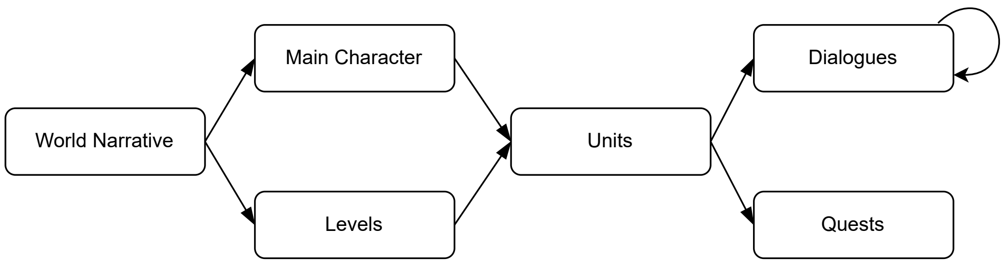


Figure 3.2 API requests structure.

Figure 3.2 demonstrates the structure of API requests. Each request (except the initial World Narrative) depends on the previously generated for this world context, and the arrows here indicate the order and direction of data transfer for the requests. Many decisions on how to instruct the model are dictated by the desire to show different approaches inside one generation.

Now, we will discuss the prompts in more detail. Each prompt has a similar structure. Firstly, there is almost the same contextual instruction **#Setting** for all request blocks that contains the following text:

```
"#Setting: \n You are a creative assistant for creating
   textual game content. You need to follow instructions
   while being creative and artistic. \n"
```

Listing 3.1 Setting instruction content

A few different versions of this were tested, but writing at least something about the context of the task helps aligning the model response. This is happening due to the sequential processing, GPT processes the prompt from start to finish and what is written earlier affect how the rest of the prompt is interpreted.

Then, there is an **#Instructions** block that contains a description of what is expected to be generated and an interpretation of referenced for the task data. The instructions given here are generally thorough enough, and some information might be repeated several times.

The optional block **#Constraints** contains consecutive numbered short instructions containing information on what should not appear in the response or sometimes repetitions of what was already given in the previous block.

Lastly, there is a **#Structure** that is usually mentioned earlier in the prompt. From practice, the structure can be provided in any part of the prompt, and the position does not affect the result noticeably. This block contains the desired structure of the response for each query.

It is also worth explaining that the Chat Completion API specifically was used in the experiment. While it is an interface that is implied to be primarily used for chat applications, it still can be used for completion tasks. The main difference between the used approach and Completion APIs is that request Chat Completion can have several messages with assigned roles. For the experiment, all the instructions described above are marked with a role system, which is usually used for that and a simple message from the user role that implies that the model should start generating the response. OpenAI themselves highlight that Completion API soon will be deprecated, and Chat Completion can be used instead [50]. The Backbone of each API request can be seen in listing 3.2. We will use temperature equals to 1.0 as a default value for every query.

```
response = openai.chat.completions.create(
    model="gpt-3.5-turbo-0125",
    temperature=1.0,
    max_tokens=4096,
    response_format={"type": "json_object"},
    messages=[
        {
            "role": "system",
            "content": "#Setting: \n "
                       "#Instructions: \n "
                       "#Constraints: \n"
                       "#Structure: \n"
        },
        {
            "role": "user",
            "content": "Start generating."
        }
    ]
)
```

Listing 3.2 API request structure

Now we will discuss details about every query. Details about mentioned entities can be found in section 2.1.

- **World Narrative.** The first thing generated is the narrative data, which includes the story type, name of the world, story, antagonist unit group and messages shown when the game starts, the game is lost, and the game is won. Story type here defines if the protagonist is evil, good or neutral, which immediately align GPT to make certain choices based on that factor. A villain group is generated in the context of the story.

  If there are any other generated worlds, it restricts GPT from generating ones with the same name. This technique is useful here and in a number of other particular cases, however, it should not be the only way we make models generate various results, because the expected behaviour is that even if there are no prior models created, GPT should output somewhat unique results.

- **Main Character.** This query generates the playable character's personality, name, class, race, occupation, and backstory. It also generates a protagonist unit group, which units are instructed to depend mostly on the protagonist's description and backstory. The protagonist is placed in the generated context of the world and its story.

- **Levels.** It generates a specified number of levels for the experiment, which is chosen to be 3. Levels are placed in the world's setting. Characteristics generated for each level are name, level description, and terrain chosen from the list of available. Prompt constraints are choosing the same walkable terrain for two different levels.

- **Unit Groups.** Here, all world enemy unit groups are generated. They are generated based on the world story and, most importantly, on the level and its environment. Each group has a name and up to 3 units. For each unit, the model is instructed to choose three unit attributes from the given database of units(discussed in section 3.3), a characteristic name based on these attributes, an artistic name and a class based on LLM knowledge about fantasy creatures, and a power level based on the instruction given in a prompt.

- **Quests.** This query is given all the previous information to consider: who is given the quest, the world setting these quests exist in, and the levels and units that are potential targets of the quest. Apart from that, there are specific instructions for possible targets, rewards, and tasks. GPT is instructed to have a quest to progress to the next level and the final quest to finish the game, which is only to defeat an antagonist.

- **Dialogues.** The dialogues from the Unit Groups query results are generated for each world enemy and antagonist. Dialogue is generated in a separate query for each enemy. The generation of the dialogue starts with taking the first unit in the group and generating personality trait ratings based on the LLM's knowledge of this fictional character and referring to the fact that this unit represents the whole enemy group. After that, based on these traits, the dialogue tree with the given parameters is generated. One of the instructions given to a model is to always generate at least one possible outcome of the dialogue for each outcome type.

3.3 Problem-Solving Examples

During the process of creating and iteratively developing the prompts, a significant number of techniques were tested, and different observations were made. Here, some of these observations will be provided along with the problems that were attempted to be resolved:

- **Mutating the input parameters.** One of the most common problems encountered while using OpenAI GPT API in the context of this thesis experiment was the task of choosing the content for the game from the given list of available assets. Despite instructions that it should take the exact string, it was often slightly changing it or coming up with completely new options.

  As the first response to this problem, the verification algorithm was implemented. It checks the input during the generation process in an attempt to find a close enough option or completely redo the query even if one unsolvable case is found. At that time, at least one generated unit asset name was broken in more than half of the generated worlds. While the verification algorithm solved the issue occasionally, the correctly generated world was sometimes received only after five attempts.

Another approach was to change the names in a list of options given to LLM by adding a numerical identification(id) to each string. As an example, instead of string *"deep_elf_mage"* representing a unit, we used *"90_deep_elf_mage"*. Adding an id to a string makes it a more easily recognisable pattern for an LLM. The model learns that the correct option has an id and it will more likely rely on the given data rather than its general knowledge. However, this approach is not well studied and adding symbols to the string might reduce the impact of these string words on a model decision process.

As a result, the most effective solution we found was to refactor the structure of the unit database. In this version, each unit is characterised by three **attributes** and stored in JSON format here is a fragment of this unit database:

```
..."undead": {
    "bog_body": [""],
    "death_cob": [""],
    "ghost": ["dragon", "eater", "flayed", "missing",
        "stalker", "void", "warrior"],...
```

Listing 3.3 Unit database fragment.

In this example sample of the database of units, the first attribute is the "undead", which is the outer JSON object. The second attributes are JSON arrays "bog_body", "death_cob", and "ghost". The third attribute is a JSON string, like "dragon", "eater", or "flayed". If there is only one unit for the second attribute, this unit does not have a third attribute. Full text of the database of units can be found in the Attachment A.2.

Using this structure and a modified query has shown the most consistent results. Providing a database in such a format makes LLM choose unit attributes sequentially, like traversing a tree. This hierarchy helps to narrow down choices based on parent categories. Numerical evaluation of this aspect of the result can be seen in section 4.3.2.

- **Mentioning features not existing in the game.** This issue means that quests, descriptions or dialogues may contain mentions of objects or mechanics that are not implemented in the game. While this does not directly affect the functionality of the game, it has a negative impact on the player's perception. Usually, such problems effect might be reduced by fine tuning, which is not a focus of this thesis. Techniques that were used are **contextual prompting**, which means that we put the GPT in the context of the game, instructing it to follow a specific role and consider certain restrictions and **iterative prompting**, which is a process of changing certain aspects of the prompt, like structure or word choice, in an attempt to improve the response quality.

4 Results

In this chapter, we will draw conclusions about the experiment and evaluate the data from multiple perspectives. First, we will discuss the result from the author's perspective, identifying what worked as expected and what could be done better. After that, we will observe the Data Set used for the evaluation and consider several selected metrics to assess the quality of the algorithm. As the last step, we will assess the game from the player's point of view.

4.1 Subjective Evaluation

The goal of this thesis was to develop an algorithm that generates Game World using OpenAI GPT API. This Game World is then used for the game created for this thesis. The game should be possible to complete and have other characteristics described in the chapter 2. The amount of content in the game that was generated should be sufficient and frequently interacted with.

From the author's perspective, the desired result was reached. The algorithm of interaction with OpenAI GPT is written in Python and C# and can be launched directly from the game as described in section 2.6.

The game GPT JRPG serves its intended purpose of utilising the content generated by LLM and being largely influenced by the choices it makes. The game usually has a clear goal derived from the Game World story, and with certain temperature 1.2.3 values, different Game Worlds are usually distinct in story, characters and other generated or partially generated game entities.

However, the algorithm and the game have several flaws. Firstly, GPT somitemies produces results that are not expected by the game and, therefore, not parsable by the mechanism of asset selection (explained in section 2.5). Prompt Engineering is a process of gradual response quality improvement, although it is difficult to evaluate its influence. While some parts of the algorithm are working as intended and the connection between the prompting approach can be identified, some things, like using generated personality traits, seemingly did not show much benefit in use.

Secondly, some systems in the GPT JRPG are underdeveloped. The main issue here is the lack of graphical assets made in one art style taken from freely available sources and diversity in in-game content, namely items, quests, endings, etc.

4.2 Data set

Here, we will discuss the Data Set used for the following evaluation. For generating a data set, we will use two different models, namely GPT-3.5 turbo and GPT-4o, and temperature (see section 1.2.3) values 0.6 and 1.2. These temperature values were chosen based on the conclusions made in the paper by Agarwal et al. [30] and experimenting with setting different temperatures for the created algorithm. Making a temperature less than 0.6 usually leads to very monotonous and similar results, and going above 1.2 leads to an enormous amount of hallucinations and errors.

Generating Dialogues is the most expensive and time-consuming part of the request. It also does not produce game crash-sensitive content, so that it won't be generated for this data set. We will discuss the dialogues separately.

Model	Temperature	Worlds	Units	Units per World
GPT-3.5 turbo	**0.6**	15	489	32.6
	1.2	15	437	29.1
GPT-4o	**0.6**	5	210	42
	1.2	5	195	39

Table 4.1 Data set used for the evaluation.

Table 4.1 shows the amount of data used for different OpenaAI API parameters used in the evaluation. We generated 15 Game Worlds for both temperature parameters for the model GPT-3.5 turbo. While the expected number of Units, according to the instruction given in the prompt, should be between 39 and 42, we can see that for both temperature parameters, it revolves around 30. The number of Generated Worlds that we will use is not enough for a reliable statistical evaluation but generating a data set of substantial size is both time-consuming and costs a significant amount of money on the API.

After looking into the data, we can observe that the number of generated units per Game World for this model lies in the range from 20 to about 40. There might be an issue with the limited number of output tokens, but in this case, there would not be such a significant spread in the number of units between worlds. So, we are assuming that it is happening due to the internal mechanism of the model or unclear instructions that are sometimes misunderstood.

On the other hand, we can see that the number of units per Game World for the GPT-4o is within the expected range. It is a bit lower for temperature 1.2 due to the hallucinations, which corrupted some data. We will discuss it more in the next section. We only generated 5 Game Worlds for each temperature here due to the price (it costs 10 times more per token than GPT-3.5 turbo and, as we stated, generates more data, and as a result, it costs around 14 times more per Game World) and that it still has enough amount of Units in order to be able to draw some general conclusions.

4.3 Succes rate

Here, we will evaluate the performance of the algorithm by observing the generated Game Worlds using the following metrics:

1. **Validity.** The Game World is considered Valid if it is parsable by the GPT JRPG internal algorithms and the game can be finished, which means the player is able to see the final game message. It will be verified by checking that each Game World can be launched in the game and that the main quests are completable (more about how the quest system works can be found in the section 2.1). Due to the low number of Game Worlds in the evaluation, we cannot draw very reliable conclusions, but we will still discuss specific cases when the Game World is corrupted and possible causes of that.

2. **Correctness.** The Correctness is the percentage of generated asset names by GPT that corresponds to the assets in a list it was supposed to choose from. This metric evaluates the Correctness of the assigned unit attributes (discussed in section 3.3), unit class, and power level 2.3. It should be noted that these issues are processed in the game and expected not to cause game crashes.

4.3.1 Validity

Let us first discuss the Validity metric. Preemptively, I would like to note that no pattern was found in terms of Validity between different API parameters. Thus, we will discuss them alternately without focusing on comparisons.

Model	Temperature	Worlds	Valid	Invalidity Causes
GPT-3.5 turbo	**0.6**	15	13	main quest
	1.2	15	14	undergenerated world
GPT-4o	**0.6**	5	5	
	1.2	5	3	hallucinations

Table 4.2 Validity evaluation for the generated Data Set and causes of Invalidity for different API parameters.

In the table 4.2, we can see that for the model GPT-3.5 turbo with temperature set to 0.6, 13 out of 15 Game Worlds are valid. In both cases, Invalidity was caused by the fact that the main quest objective on the first level was the unit from the different level which cannot be accessed without completing this quest. I would note here that it appeared to be a common issue for the side quests, especially for the GPT-3.5 turbo model. Around 8 out of 15 Game Worlds for both temperature values contain such quests. It doesn't make the Game World invalid, but it is still a significant gameplay flaw. It also indicates that the chosen approach in generating quests might be defective.

For the same model with a temperature set to 1.2, there is only one invalid world. It cannot be processed by the game because JSON files contain empty fields for the second and the third levels, therefore, the game cannot be finished. While we cannot make definitive conclusions about it due to the small data set, we expect to receive more structural issues from the requests with higher temperatures.

Now, let us discuss the performance of the model GPT-4o. We will start with observing results received with a temperature set to 1.2. Interestingly, responses received with such parameters are more hallucinated than with the older, less capable model. Here, we can see the sample of the response part when the response started to receive hallucinations:

```
...{
          "groupName": " Dormant_pushed as lifted
             Postererts of gigantic Seculence",
          "units \u0430\u0432\u0442\u043e({' Explor
             certain values ideal_prelolajia of water
             Blat extra_MASK myst Other tension detail
             lake visual Produce approachable attit
             reversible igual nuances envelop Morr.
             boldanlage arrangement": "units\"\ngoe
             tissues_fact prescribed apl CAce refined She
             'int_dim size unseen decorative Mith ",
          "health...: []
        }
```

Listing 4.1 Corrupted Units Data file sample.

While only 2 worlds are invalid, the content generated for others will hardly be used in real games. As the example for one of the valid worlds, it generated such a game over message:

```
{..."gameOverMessage": "Recoil in despair, defeated unto
   raising the fists of your folly! Balance shall
   reassert. Olymorr and his Guardians invoke as veils
   shred illusions not pleasant. Darkness, returning to
   silent embrace, for now abides elsewhere."...}
```

Listing 4.2 Hallucinations in game over message.

Finally, the model GPT-4o with temperature set to 0.6 had almost no issues, and even if we look at quests, only in 1 Game World side quest were they incompletable. Based on the Validity metric and our observations, these parameters are preferred compared to others.

4.3.2 Correctness

Now, let us observe the Correctness of the Game Worlds from the Data Set. We will compare this metric for different API parameters and analyse problematic segments.

Model	Temperature	Units	Correct	Correctness
GPT-3.5 turbo	**0.6**	489	469	95.9%
	1.2	437	380	87.0%
GPT-4o	**0.6**	210	208	99.0%
	1.2	195	170	87.2%

Table 4.3 Correctness for different API parameters.

The table 4.3 shows us a few noteworthy points, namely, we might make the conclusion that there is a direct correlation between the temperature and the calculated Correctness. We receive a lot more consistent results using a lower temperature, especially for the GPT-4o, which shows 99.0% Correctness with only 2 problematic Units. However, we might also observe that for a temperature equal to 0.6, we don't have any advantages in using the GPT-4o over the GPT-3.5 turbo with 87.2% and 87.0% percentage of correct Units, respectively.

At the beginning of the section, we mentioned what can cause the Incorrectness of the Unit. We will only take Units that are Incorrect and analyse what is the most frequent reason for that for different API parameters.

API Param.	Total	Power	Class	Attr. 1	Attr. 2	Attr. 3
GPT-3.5 turbo, temp=0.6	**20**	0(0%)	0(0%)	0(0%)	6(30.0%)	14(70.0%)
GPT-3.5 turbo, temp=1.2	**66**	3(5.3%)	2(3.5%)	1(1.8%)	27(47.4%)	26(45.6%)
GPT-4o, temp=0.6	**2**	0(0%)	0(0%)	0(0%)	1(50.0%)	1(50.0%)
GPT-4o, temp=1.2	**25**	4(16%)	8(32%)	2(8%)	9(36.0%)	12(48.0%)

Table 4.4 Details on what caused the Incorrectness of Units for different API parameters.

In the table 4.4, we can see that Units generated for the temperature 0.6 do not have issues with the chosen for them power level and class (see section 2.3), as well as the first attribute (see section 3.3). Listed entities are chosen by GPT from the list and using lower temperature causes it to not invent not existing in the given input options. It is different for the second and the third attribute because GPT instructed to narrow down the choice to the certain JSON object in the unit database (Attachment A.2), and it is sometimes choosing these attributes from the neighbour objects, which is not intended and such a Unit combination of attributes is not recognized by the GPT JRPG.

For the higher (1.2) temperature, the situation with the second and the third attributes is only getting worse, and on top of that, the model starts to distort existing or come up with completely new power levels, classes, and, less often, first attributes. As a few examples, GPT-3.5 turbo changes existing ”mighty” to ”might” while GPT-4o is creating entirely new like ”deadly”. GPT 4o is also sometimes producing nonsense results, as shown below:

```
{..."units": [
      {
        "firstAttribute": "hawassrngeestyle",
        "secondAttribute": "imp",
        "thirdAttribute": "shadow",
        "characteristicName": "Dark Imp",
        "artisticName": "Grimalker, the Sneak",
        "powerLevel": "mighty",
        "unitClass": "trickster"
      },...}
```

Listing 4.3 Pointwise hallucinations in the Unit JSON object.

From the context, it is clear that the first attribute should have been ”hell”, but in the case of this particular query (same as for the listing 4.2), it produced these unexpected pointwise hallucinations. It is also worth noting that the wrong first attribute makes checking the following one impossible, and they are not counted in the table 4.4.

One of the ways to improve the Correctness of the Game World might be to increase the number of prompts in the chain (we discussed the technique named chain prompting in section 3.1.1). We can potentially use separate queries for each unit and, inside these queries, have a conversation with OpenAI GPT API to alternately ask it to choose attributes. During these conversations, we should provide isolated parts of the unit database so it has no chance to choose something that does not belong to the previously chosen attributes. However, it would result in at least 30 to 40 queries in which we would need to include the information about all the previously generated units and also implement an additional conversation with the OpenAI GPT API. This would result in a dramatic increase in generation price.

In conclusion, after observing the Correctness metric for different API parameters, we can see that the model GPT-4o with the temperature set to 0.6 is the better choice for our application.

4.4 Game Quality

In this section, we will observe the game from the player’s perspective. Namely, how the content-choosing mechanism is diverse for different API parameters, assess the dialogue quality and make a few general observations about the game.

First, we will again look at the result of the data set evaluation. We can see that for both metrics, the best performance has shown the API queries with lower (0.6) temperature and used the GPT-4o model. The second best (at least for the Correctness metric) was the result shown by GPT-3.5 turbo with the same temperature.

However, if look at some other parameters, like Game World Name or diversity in chosen units, we will see the expected less diverse and artistic response for the lower temperature. In the case of GPT-3.5 turbo, it sometimes produced entire levels with identical Unit groups for the temperature set to 0.6. On the other hand, while we cannot state that for the same model with a higher temperature, we received very rich results, this behaviour was not met for any other API parameters. Considering these observations, we cannot make a strong conclusion on which temperature to use for the model GPT-3.5 turbo and it would require conducting user research.

In the case of the model GPT-4o, it is more obvious since both models made diverse choices and produced diverse enough, from our perspective, results. Due to the enormous hallucinations when the temperature is set to 1.2, we can state that the optimal choice here is the temperature set to 0.6.

4.4.1 Dialogues

Here, we will assess the dialogues from a few different perspectives.

First of all, based on the observations made in the previous sections, we decided to use a temperature set to 1.0. This is mainly because we wanted to attempt to avoid hallucinations when using the model GPT-4o and we are lowering it also for GPT-3.5 turbo for simplicity. We won't observe dialogues generated with the lower temperature, since we want to make an accent on assessing the creativity of these dialogues.

We will take one of the Game Worlds, generated with the model GPT4o, and temperature equals 0.6, from the data set. We will generate dialogues using previously discussed parameters for the same world.

We will compare the dialogues with the following unit from this world and draw a few conclusions:

```
{..."unit": {"firstAttribute": "beast",
      "secondAttribute": "troll",
      "thirdAttribute": "forest",
      "characteristicName": "Forest Troll",
      "artisticName": "Grumok",
      "powerLevel": "strong"},...}
```

Listing 4.4 Unit JSON object

And in listings 4.5 and 4.6, we can see the initial part of the dialogue with Forest Troll Grumok.

```
{..."dialogueTree": {
    "enemyPhrase": "You dare enter our domain, puny
      creature. What brings you here?",
    "playerCharacterAnswers": [
      {
        "playerOption": "I seek to challenge the might of
           the forest trolls!",
        "enemyAnswer": "Hah! A foolish decision. Prepare
          to face the consequences of your arrogance.",
        "playerCharacterAnswers": [
          {
            "playerOption": "I am ready to face whatever
              challenges you throw at me!",
            "enemyAnswer": "Brave words, but courage
              alone won't save you from the wrath of the
               forest. Prepare for a battle you won't
              forget.",,...}
```

Listing 4.5 Fragment of the dialogue generated with the model GPT-3.5 turbo and temperature 1.0.

```
{..."dialogueTree": {
    "enemyPhrase": "You stumble into Grumok's domain,
      puny creature! Speak quick, or be eaten slow!",
    "playerCharacterAnswers": [
      {
        "playerOption": "I seek the dark artifacts within
           this forest. Lend me your strength, and I
          shall share my power with you.",
        "enemyAnswer": "Grumok not care for puny
          artifacts. Grumok care for strength. Show you
          strong, then maybe talk.",
        "playerCharacterAnswers": [
          {
            "playerOption": "Very well, I'll prove my
              strength. Let's wrestle, beast!",
            "enemyAnswer": "Grumok like wrestle! But if
              you lose, Grumok feast on you!",...}
```

Listing 4.6 Fragment of the dialogue generated with the model GPT-4o and temperature 1.0.

The first observation we can make here is that GPT-4o offer a dialogue that is trying to capture the fantasy nature of the Troll, which lies in not being a native human language speaker. It often uses short, simple phrases referring to themselves in the third person and displaying signs of aggression.

At the same time, GPT-3.5 turbo gives us a decent but pretty generic dialogue. Troll still mentions the environment it lives in, but if we look at any other dialogue produced by this model, we will see very similar phrases and formulations no matter which nature is the Unit we speak to.

Here is the fragment of the Dialogue prompt with specific instructions about the speaking style:

```
"Be radical about changing enemy speaking style. Base it
   mainly on the most extremely low or high metric, don't
    hesitate to be very rude or very friendly, very
   thorough or very simple, etc."
"Derive specific speaking artifacts, inherent sounds,
   literature accents, idioms or slang based on your
   knowledge about creatures in a fantasy world."
```

Listing 4.7 Fragment of the prompt for Dialogues.

In our view, GPT-4o fulfils these instructions in this particular case. At the same time, GPT-3.5 turbo rarely implements any linguistic features.

In conclusion, the most prominent combinations of API parameters from our observations is to use the GPT4o model with lower (0.6) values for all requests (the structure of API queries is discussed in the section 3.2), but dialogues for which the temperature was set to 1.0. There might be a better and more complex combination of API parameters that will result in better responses to our queries. However, it is not in the scope of our thesis due to the high API usage cost and low practical use.

5 Future Work

The implemented approach might be considered adequate for given conditions but is far from being universal. Instead of trying to make it more versatile at the moment, we can say that to develop this project and further study the potential of using LLMs for PCG, it is worth focusing on iteratively improving the interaction algorithm with the model and increasing the scale and quality of the game. The desirable outcome might be a finished computer game that would be appreciated by the audience.

- **Game improvement.** One way to improve the experiment is to increase the game's scope. Currently, the game has a very restricted number of mechanics and features. As a result, the LLM is not quite as varied.

  Developing the idea that the LLM can choose various options from those offered, the following features could also be added to the game architecture. One of them is to introduce universal building blocks that represent different game mechanics and are combined together to form an adventure.

  Such event blocks might be represented as a dialogue with an NPC, finding an item, solving a puzzle, engaging in battle, taking a quest, inspecting the area, etc. Some of these blocks might not be combined due to their nature, so there definitely should be a filler block, like a universally available puzzle. Event blocks can occur continuously or be in a waiting mode until an expected action occurs, which can be visualised as a quest waiting to be completed.

  The game should provide a system that supports such an idea. It should verify that the desired goal of the block can be completed and that all entities mentioned in this block are currently available in the game. The system resembles the one described in a reference in the section 1.3.1 paper can be potential. It would store the information about all the possible events and items and will interact with LLM to process event blocks, detect and resolve event dependency cycles and bind everything to an event timeline.

  There might be different issues to resolve with such an architecture. However, this idea will not be expanded further and is left to be tested in future experiments.

- **Prompt Engineering.** One of the approaches on how the quality of the received from LLM data can be improved is by using fine-tuning. Namely, a data augmentation technique called bootstrapping. This approach takes data produced by the LLM and uses it to fine-tune itself further. Not all the data produced by the model might be suitable for fine-tuning. In the paper by Ulmer et al. [51], the automated filter was used to determine which data should be used. The schema of how this process was implemented in their experiment is shown in figure 5.1. For an experiment similar to one conducted for the thesis, we may use human supervision to choose the most appropriate data which might be modified. This approach would still need enough human-created data for initial fine-tuning to avoid artificial bias in the model.

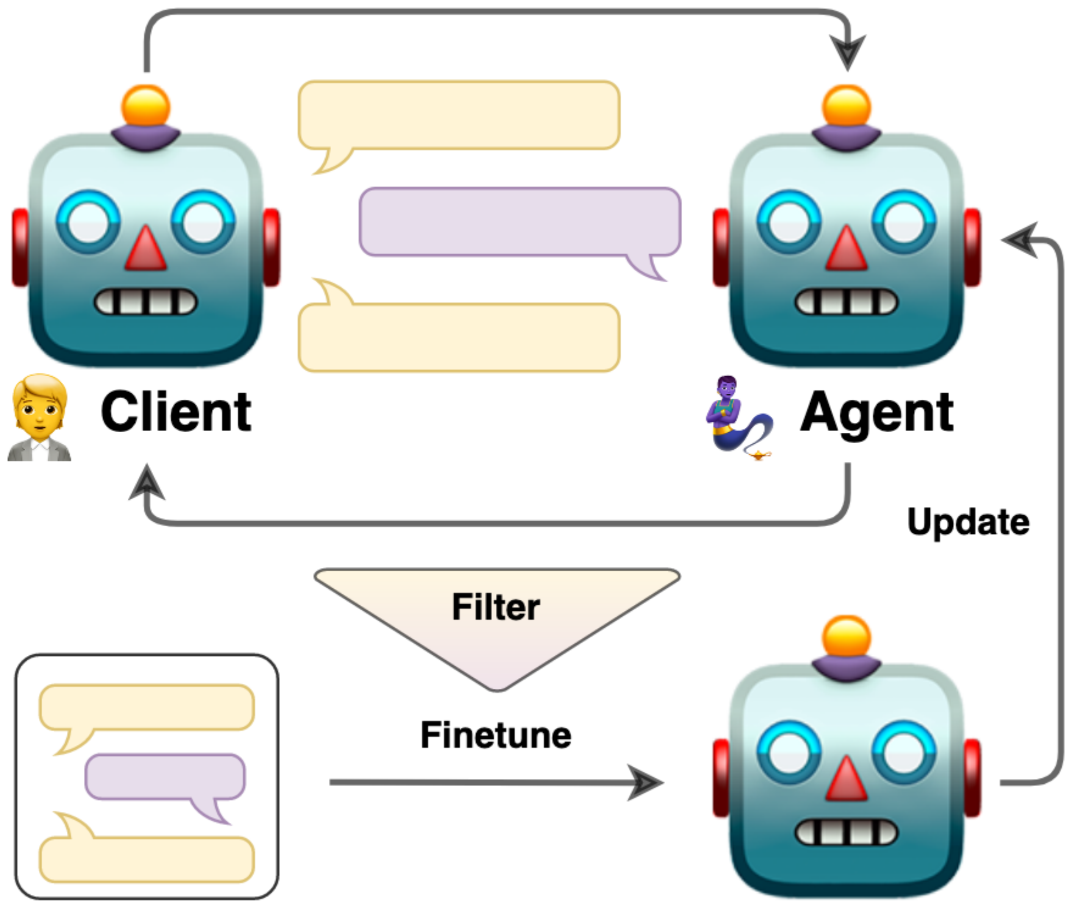


Figure 5.1 Bootstraping schema.
Source: Ulmer et al. [51]

Using fine-tuning might benefit the precision of the model and consistency of responses but negatively affect the variability of answers.

- **User research.** What has not been done as part of the thesis but is definitely worth doing for further work is conducting a survey that might have included the following as the core survey.
  - **First part of the survey.** Evaluating metrics of different parts of the content overall and separately: dialogues, names, quests, and descriptions. This includes the text's quality from the survey respondent's personal perspective, clarity of what is being disputed in dialogues or explained in quests, how cohere the text is and if it adheres to the same style, if the content resembles what was generated before or feels authentic, etc.

    This survey should conduct A/B testing using both generated and human-created content. The results for both cases will be compared, and the weak points of the generated content will be better identified.
  - **Second part of the survey.** One of the possible improvements of the experiment that is also connected to user research is adding input

to the prompt that is being used in API requests. Namely, inputting several parameters, such as the world setting theme and personality traits of the protagonist or antagonist. Along with different numerical parameters, such as number of levels, enemies, dialogue tree depth, etc. While numerical parameters can be restricted by minimum and maximum available values, fields that allow textual input should be supplemented with instructions on which input is expected. It can also be supervised by a separate API query that can verify that input is appropriate and, if not, give a corresponding warning that it should be rewritten. This can potentially provide a unique experience for players who will be able to go through their own vaguely described fantasy story.

  After that, respondents can evaluate how accurately the result resembles their expectations about the input. If they can claim that the Game World generated with their instructions is better or worse overall than that given to them in the first part of the survey.

  Based on the second part of the survey, we can evaluate whether allowing the player to impact the generation process improves the overall experience and identify vulnerabilities in adding player input to the prompt algorithm.

- **Multimodal LLMs.** Another prominent for the PCG system is the Multimodal Large Language Models (MLLMs). MLLMs are capable of not only generating text but also working with other modalities, such as audio, video or images. This might allow us to further extend the Game World scope and make the used approach more versatile.

  From our experience, the current state of these systems might not be of the desired standard. We tried to use ChatGPT with integrated image generation DALL · E model [52] to generate assets for the GPT JRPG and, despite all the effort, were not able to receive consistent results. Mainly because it struggles with PNG transparency and doesn't support generating pixel art. However, MLLMs have a great potential for PCG and are definitely worth studying in future.

Conclusion

Large Language Models are one of the most significant and valuable developments of recent decades. They have already proven their value in many different fields, and in this thesis, we aimed to develop our approach to use them for Procedural Content Generation for games. To do that, we explored various existing approaches for interacting with LLMs and relevant to our thesis techniques from PCG.

As a result, we implemented an application that interacts with the OpenAI API and receives structured content as an output. We also created a game in the Japanese Role-Playing Game genre that uses this content to attempt to provide players with replayability and a diverse narrative. While the game currently meets the quality expectations and can successfully represent the generated content, there is definitely room for improvement both in terms of implementing a more reliable and flexible internal architecture and upgrading the visual aesthetics.

It is a non-trivial task to assess the quality of the interaction with LLMs. However, we tried to conduct a comprehensible analysis of the received results and made a few observations. In this thesis, we focused on implementing the algorithm for generating content for a particular game, and there are quite some directions for additional research, both in advancing a created application and developing a more universal approach. We hope that the conclusions made in this work might be useful in the following studies and for developers who would like to try a similar approach to generate content.

We see great potential in using LLMs for PCG because they could significantly improve the variability of the generated content in games and reduce the cost of development, which is particularly useful for small and indie developers because of LLMs' accessibility. On top of that, the unpredictable nature of LLMs may result in them being used to invent new game mechanics or entire genres. We hope that LLMs will continue to evolve in future and can become an even more accessible and useful tool that can be used to enhance the game development process.

Bibliography

1. Shaker, Noor; Togelius, Julian; Nelson, Mark J. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer, 2016. Available from DOI: 10.1007/978-3-319-42716-4.
2. *Rogue (video game) Wikipedia Page* [online]. [visited on 2024-07-01]. Available from: https://en.wikipedia.org/wiki/Rogue_(video_game).
3. *Berlin Interpretation - RogueBasin Wiki* [online]. RogueBasin. [visited on 2024-07-01]. Available from: https://www.roguebasin.com/index.php/Berlin_Interpretation.
4. *No Man's Sky* [online]. [visited on 2024-07-03]. Available from: https://www.nomanssky.com/.
5. *Diablo II: Resurrected* [online]. [visited on 2024-07-03]. Available from: https://diablo2.blizzard.com/.
6. Batchelor, James. PONG. In: *The Best Non-Violent Video Games.* United States: Pen & Sword Books Limited, 2023, pp. 25–26. ISBN 9781399084925.
7. Barnabé, Fanny; Bjarnason, Nökkvi Jarl; Blom, Joleen; Campana, Andrew; Huber, William; Johnson, Daniel; Koyama, Yuhsuke; Mineau-Murray, Loïc; Mondelli, Frank; Nakagawa, Daichi, et al. *Japanese role-playing games: genre, representation, and liminality in the JRPG.* Rowman & Littlefield, 2022.
8. *Baten Kaitos I & II HD Remaster | Official Website* [online]. Bandai Namco. [visited on 2024-07-15]. Available from: https://en.bandainamcoent.eu/baten-kaitos/baten-kaitos-i-ii-hd-remaster.
9. *YO-KAI WATCH - Nintendo 3DS - Games - Nintendo* [online]. Nintendo. [visited on 2024-07-15]. Available from: https://www.nintendo.com/au/games/nintendo-3ds/yo-kai-watch/.
10. *Sea of Stars* [online]. [visited on 2024-07-03]. Available from: https://seaofstarsgame.co/.
11. *Introducing GPT-4o and more tools to ChatGPT free users* [online]. OpenAI. [visited on 2024-07-07]. Available from: https://openai.com/index/gpt-4o-and-more-tools-to-chatgpt-free/.
12. Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz; Polosukhin, Illia. Attention is all you need. *Advances in neural information processing systems.* 2017, vol. 30. Available from DOI: 10.5555/3295222.3295349.
13. Mohebi, Laila. Empowering learners with ChatGPT: insights from a systematic literature exploration. *Discover Education.* 2024, vol. 3, no. 1, p. 36. Available from DOI: 10.1007/s44217-024-00120-y.
14. Buruk, Oğuz 'Oz'. Academic Writing with GPT-3.5 (ChatGPT): Reflections on Practices, Efficacy and Transparency. In: Tampere, Finland: Association for Computing Machinery, 2023, pp. 144–153. Mindtrek '23. ISBN 9798400708749. Available from DOI: 10.1145/3616961.3616992.

15. Li, Mingchen; Blaes, Anne; Johnson, Steven; Liu, Hongfang; Xu, Hua; Zhang, Rui. CancerLLM: A Large Language Model in Cancer Domain. *arXiv preprint arXiv:2406.10459*. 2024. Available from DOI: `10.48550/arXiv.2406.10459`.

16. *OpenAI Platform. Documentation* [online]. OpenAI. [visited on 2024-07-07]. Available from: `https://platform.openai.com/docs/overview`.

17. Lelli, Francesco. *Dwarf Fortress' creator on how he's 42 % towards simulating existence* [online]. [visited on 2024-07-06]. Available from: `https://francescolelli.info/tutorial/neural-networks-a-collection-of-youtube-videos-for-learning-the-basics/`.

18. Lin, Tianyang; Wang, Yuxin; Liu, Xiangyang; Qiu, Xipeng. A survey of transformers. *AI Open*. 2022, vol. 3, pp. 111–132. ISSN 2666-6510. Available from DOI: `https://doi.org/10.1016/j.aiopen.2022.10.001`.

19. Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *North American Chapter of the Association for Computational Linguistics*. 2019. Available also from: `https://api.semanticscholar.org/CorpusID:52967399`.

20. Radford, Alec; Narasimhan, Karthik. Improving Language Understanding by Generative Pre-Training. In: 2018. Available also from: `https://api.semanticscholar.org/CorpusID:49313245`.

21. Czinczoll, Tamara; Hönes, Christoph; Schall, Maximilian; Melo, Gerard de. NextLevelBERT: Investigating Masked Language Modeling with Higher-Level Representations for Long Documents. *arXiv:2402.17682*. 2024. Available from DOI: `10.5555/3295222.3295349`.

22. Kaplan, Jared; McCandlish, Sam; Henighan, Tom; Brown, Tom B; Chess, Benjamin; Child, Rewon; Gray, Scott; Radford, Alec; Wu, Jeffrey; Amodei, Dario. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*. 2020. Available from DOI: `10.48550/arXiv.2001.08361`.

23. Simon, Julien. *Large Language Models: A New Moore's Law?* [online]. Hugging Face. [visited on 2024-07-06]. Available from: `https://huggingface.co/blog/large-language-models`.

24. Zhao, Wei; He, Ting; Xu, Li. Enhancing Local Dependencies for Transformer-Based Text-to-Speech via Hybrid Lightweight Convolution. *IEEE Access*. 2021, vol. 9, pp. 42762–42770. Available from DOI: `10.1109/ACCESS.2021.3065736`.

25. *OpenAI Website* [online]. OpenAI. [visited on 2024-07-07]. Available from: `https://openai.com/`.

26. *Mistral AI Website* [online]. Mistral AI. [visited on 2024-07-07]. Available from: `https://mistral.ai/`.

27. *Gemini Website* [online]. Google AI. [visited on 2024-07-07]. Available from: `https://gemini.google.com/`.

28. *Cohere Website* [online]. Cohere Inc. [visited on 2024-07-07]. Available from: `https://cohere.com/`.

29. *Llama Website* [online]. Meta AI. [visited on 2024-07-07]. Available from: `https://llama.meta.com/`.

30. AGARWAL, Arav; MITTAL, Karthik; DOYLE, Aidan; SRIDHAR, Pragnya; WAN, Zipiao; DOUGHTY, Jacob Arthur; SAVELKA, Jaromir; SAKR, Majd. Understanding the Role of Temperature in Diverse Question Generation by GPT-4. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. Portland, OR, USA: Association for Computing Machinery, 2024, pp. 1550–1551. SIGCSE 2024. ISBN 9798400704246. Available from DOI: `10.1145/3626253.3635608`.

31. GUMIN, Maxim. *Wave Function Collapse* [online]. [visited on 2024-07-04]. Available from: `https://github.com/mxgmn/WaveFunctionCollapse`.

32. YASWINSKI, Matthew; CHELLADURAI, Jeyaprakash; BAROT, Shivani. A Virtual Reality Game Utilizing L-Systems for Dynamic Level Generation. *Journal of Advances in Information Technology*. 2024, vol. 15, no. 2. Available from DOI: `10.12720/jait.15.2.276-280`.

33. MARTIN, Glenn; HUGHES, Charles; SCHATZ, Sae; NICHOLSON, Denise. The use of functional L-systems for scenario generation in serious games. 2010, p. 6. ISBN 978-1-4503-0023-0. Available from DOI: `10.1145/1814256.1814262`.

34. *World generation* [online]. Minecraft Wiki. [visited on 2024-07-04]. Available from: `https://minecraft.wiki/w/World_generation`.

35. BALINT, J. Timothy; BIDARRA, Rafael. Procedural Generation of Narrative Worlds. *IEEE Transactions on Games*. 2023, vol. 15, no. 2, pp. 262–272. Available from DOI: `10.1109/TG.2022.3216582`.

36. *Q&A: Dissecting the development of Dwarf Fortress with creator Tarn Adams* [online]. [visited on 2024-07-03]. Available from: `https://www.gamedeveloper.com/design/q-a-dissecting-the-development-of-i-dwarf-fortress-i-with-creator-tarn-adams`.

37. SHORT, Tanya X; ADAMS, Tarn. *Dwarf Fortress, Moon Hunters, and Practices in Procedural Generation* [online]. Youtube. [visited on 2024-07-03]. Available from: `https://www.youtube.com/watch?v=v8zwPdPvN10`.

38. *Dwarf Fortress' creator on how he's 42 % towards simulating existence* [online]. PCGamer. [visited on 2024-07-03]. Available from: `https://www.pcgamer.com/dwarf-fortress-creator-on-how-hes-42-towards-simulating-existence/2/`.

39. SHORT, Tanya X; ADAMS, Tarn. *Procedural storytelling in game design*. Crc Press, 2019. Available from DOI: `10.1201/9780429488337`.

40. SHORT, Tanya X. *6 Techniques for Leveraging AI in Content Generation* [online]. Youtube. [visited on 2024-07-04]. Available from: `https://youtu.be/priaBvs441Y?si=EW7UDOmLyL11zxOx&t=623`.

41. NASIR, Muhammad U; TOGELIUS, Julian. Practical PCG Through Large Language Models. In: *2023 IEEE Conference on Games (CoG)*. 2023, pp. 1–4. Available from DOI: `10.1109/CoG57401.2023.10333197`.

42. *Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine* [online]. Unity. [visited on 2024-07-11]. Available from: https://unity.com/.

43. Dong, Qingxiu; Li, Lei; Dai, Damai; Zheng, Ce; Wu, Zhiyong; Chang, Baobao; Sun, Xu; Xu, Jingjing; Li, Lei; Sui, Zhifang. A Survey for In-context Learning. *ArXiv*. 2023, vol. abs/2301.00234. Available from DOI: 10.48550/arXiv.2301.00234.

44. White, Jules; Fu, Quchen; Hays, Sam; Sandborn, Michael; Olea, Carlos; Gilbert, Henry; Elnashar, Ashraf; Spencer-Smith, Jesse; Schmidt, Douglas C. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*. 2023. Available from DOI: 10.48550/arXiv.2302.11382.

45. Bsharat, Sondos Mahmoud; Myrzakhan, Aidar; Shen, Zhiqiang. Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4. *arXiv preprint arXiv:2312.16171*. 2023. Available from DOI: 10.48550/arXiv.2312.16171.

46. Wei, Jason; Bosma, Maarten; Zhao, Vincent Y; Guu, Kelvin; Yu, Adams Wei; Lester, Brian; Du, Nan; Dai, Andrew M; Le, Quoc V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*. 2021. Available from DOI: 10.48550/arXiv.2109.01652.

47. Brown, Tom; Mann, Benjamin; Ryder, Nick; Subbiah, Melanie; Kaplan, Jared D; Dhariwal, Prafulla; Neelakantan, Arvind; Shyam, Pranav; Sastry, Girish; Askell, Amanda, et al. Language models are few-shot learners. *Advances in neural information processing systems*. 2020, vol. 33, pp. 1877–1901. Available from DOI: 10.48550/arXiv.2005.14165.

48. Wei, Jason; Wang, Xuezhi; Schuurmans, Dale; Bosma, Maarten; Xia, Fei; Chi, Ed; Le, Quoc V; Zhou, Denny, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*. 2022, vol. 35, pp. 24824–24837. Available from DOI: 10.48550/arXiv.2201.11903.

49. Riedel, Sebastian; Kiela, Douwe; Lewis, Patrick; Piktus, Aleksandra. *Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models* [online]. Meta AI. [visited on 2024-07-08]. Available from: https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/.

50. *GPT-4 API general availability and deprecation of older models in the Completions API* [online]. OpenAI. [visited on 2024-07-07]. Available from: https://openai.com/index/gpt-4-api-general-availability/.

51. Ulmer, Dennis; Mansimov, Elman; Lin, Kaixiang; Sun, Justin; Gao, Xibin; Zhang, Yi. Bootstrapping llm-based task-oriented dialogue agents via self-talk. *arXiv preprint arXiv:2401.05033*. 2024. Available from DOI: 10.48550/arXiv.2401.05033.

52. *DALL · E 3 is now available in ChatGPT Plus and Enterprise* [online]. OpenAI. [visited on 2024-07-09]. Available from: https://platform.openai.com/docs/overview.

List of Figures

List of Tables

A Attachments

A.1 Unit classes and power levels.

```
{
  "unitClass": {
    "fighter": {
      "healthCoefficient": 0.9,
      "damageCoefficient": 1.2,
      "armourCoefficient": 0.9
    },
    "sorcerer": {
      "healthCoefficient": 0.6,
      "damageCoefficient": 1.4,
      "armourCoefficient": 1.0
    },
    "paladin": {
      "healthCoefficient": 1.6,
      "damageCoefficient": 0.7,
      "armourCoefficient": 0.7
    },
    "protector": {
      "healthCoefficient": 2.1,
      "damageCoefficient": 0.1,
      "armourCoefficient": 1.2
    },
    "bastion": {
      "healthCoefficient": 0.7,
      "damageCoefficient": 0.7,
      "armourCoefficient": 1.6
    },
    "healer": {
      "healthCoefficient": 1.0,
      "damageCoefficient": 1.0,
      "armourCoefficient": 1.0
    },
    "trickster": {
      "healthCoefficient": 1.2,
      "damageCoefficient": 1.2,
      "armourCoefficient": 0.6
    },
    "berserker": {
      "healthCoefficient": 1.5,
      "damageCoefficient": 1.5,
      "armourCoefficient": 0
    },
    "marksman": {
      "healthCoefficient": 1.0,
      "damageCoefficient": 2.0,
      "armourCoefficient": 0
    },
    "shaman": {
      "healthCoefficient": 1.3,
      "damageCoefficient": 1.3,
      "armourCoefficient": 0.4
    }
  },
  "powerLevelAttributes": {
    "feeble": {
      "health": 100,
      "damage": 15,
      "armour": 4
    },
    "weak": {
      "health": 110,
      "damage": 18,
      "armour": 5
    },
    "moderate": {
      "health": 120,
      "damage": 21,
      "armour": 6
    },
    "strong": {
      "health": 135,
      "damage": 24,
      "armour": 7
    },
    "mighty": {
      "health": 150,
      "damage": 27,
      "armour": 8
    },
    "formidable": {
      "health": 175,
      "damage": 30,
      "armour": 10
    },
    "legendary": {
      "health": 200,
      "damage": 35,
      "armour": 12
    }
  }
}
```

A.2 Units database.

```
{
  "abomination": {
    "evil_eye": [""],
    "giant_orange_brain": [""],
    "jelly": [""],
    "lab_rat": [""],
    "ooze": ["acid", "azure", "blood", "death", "plague",
        "plant", "spectral", "tentacle", "void"],
    "orb_guardian": [""],
    "pulsating_lump": [""]
  },
  "beast": {
    "mutant": [""],
    "crystalid": [""],
    "entropy_weaver": [""],
    "formicid": [""],
    "venom": [""],
    "hydra": [""],
    "hydra_lord": [""],
    "hydrataur": [""],
    "juggernaut": [""],
    "lindwurm": [""],
    "moth": [""],
    "troll": ["forest", "rock"],
    "wyvern": [""]
  },
  "birdlike": {
    "griffon": [""],
    "harpy": [""],
    "hippogriff": [""],
    "kenku_winged": [""],
    "manticore": [""],
    "quasit": [""],
    "raven": [""],
    "tengu": ["fighter", "conjurer", "reaver", "warrior"]
  },
  "hell": {
    "balrug": [""],
    "demon": ["shadow", "slave"],
    "devil": ["hairy", "red", "rotting"],
    "hell_knight": ["defender", "lord"],
    "hell_sentinel": [""],
    "imp": ["torturer", "shadow"],
    "lava_worm": [""],
    "pit_fiend": [""],
    "salamander": ["fighter", "firebrand", "mystic", "stormcaller"]
  },
  "draconid": {
    "dragon": ["eastern", "golden", "green", "ice", "iron", "quicksilver", "shadow", "storm", "swamp"],
    "drake": ["death", "fire", "forest", "mottled", "steam", "swamp"]
  },
  "primordial": {
    "elemental": ["air", "frost", "earth", "fire", "iron", "water"],
    "gargoyle": ["metal", "molten", "stone"],
    "giant": ["fire", "frost", "stone"],
    "golem": ["clay", "frost", "electric", "fire", "guardian", "iron", "stone", "toenail", "wood"],
    "lord": ["fire", "frost", "stone"],
    "phoenix": [""],
    "servitor_spellforged": [""]
  },
  "holy": {
    "angel": ["warmonger", "shieldwielder", "macewielder", "peacemaker"],
    "apys": [""],
    "centaur_ascended": [""],
    "cherub": [""],
    "daeva": ["shieldwielder", "warmonger"],
    "dragon_ascended": [""],
    "human_ascended": [""],
    "ophan": [""],
    "shedu": [""],
    "titan": ["annihilator", "brawler"]
  },
  "horde": {
    "centaur": ["archer", "fighter", "sergeant_archer", "sergeant_fighter"],
    "cyclops": ["chief", "fighter"],
    "deep_troll": ["berserker", "earth_mage", "fighter", "shaman"],
    "ettin": ["king", "warrior"],
    "hill_giant": ["annihilator", "fighter"],
    "iron_troll": ["fighter", "monk_ghost"],
    "minotaur": [""],
    "ogre": ["archimage", "fighter", "mage", "warrior"],
    "spriggan": ["berserker", "defender", "enchanter", "rider"],
    "two_headed_ogre": ["berserk", "fighter"],
    "yaktaur": ["arbalester", "chief", "fighter", "
       marksman"]
  },
  "humanoid": {
    "big_kobold": ["fighter", "sergeant"],
    "deep_dwarf": ["artificer", "berserker", "
       death_knight", "peasant"],
    "deep_elf": ["annihilator", "blademaster", "conjurer"
       , "death_mage", "defender", "demonologist", "
       fighter",
      "high_priest", "knight", "mage", "master_archer", "
         priest", "sergeant", "soldier", "sorcerer", "
         summoner"],
    "dwarf": ["chief", "warrior"],
    "elf": ["militia", "peasant"],
    "gnoll": ["defender", "fighter", "sergeant", "shaman"
       ],
    "gnome_mage": [""],
    "goblin": ["defender", "fighter"],
    "halfling": ["boy", "peasant"],
    "hobgoblin": ["defender", "fighter"],
    "human": ["archimage", "enchantress", "militia", "
       monk_ghost", "peasant", "slave", "tradesman"],
    "ironbrand_convoker": [""],
    "ironheart_preserver": [""],
    "killer_klown": ["blue", "green", "purple", "red", "
       yellow"],
    "kobold": ["aggressor", "demonologist", "peasant"],
    "orc": ["apprentice", "berserk", "druid", "guard", "
       high_priest", "knight", "necromancer",
      "peasant", "priest", "sorcerer", "summoner", "
         torturer", "warlord", "warrior", "wizard"]
  },
  "relict": {
    "anubis_guard": [""],
    "apocalypse_crab": [""],
    "boggart": [""],
    "dryad": [""],
    "faun": [""],
    "satyr": [""],
    "shapeshifter": ["artifficier", "radiant"],
    "simulacrum": ["ant", "bat", "bee", "centaur", "
       dragon", "drake", "fish", "giant",
      "hydra", "hydra_lord", "kraken", "lizard", "naga",
         "quadruped", "snake", "spider"],
    "sphinx": [""],
    "stone_ghost": [""],
    "ushabti": [""],
    "water_nymph": [""]
  },
  "serpentide": {
    "merfolk": ["aquamancer", "avatar", "defender", "
       fighter",
      "impaler", "javelineer", "peasant", "plain", "
         warrior", "witch"],
    "mermaid": [""],
    "naga": ["fighter", "guardian", "mage", "peasant", "
       ritualist", "sharpshooter", "warrior", "
       warrior_queen"],
    "serpent": ["cobra", "guardian"],
    "siren": ["singer", "witch"]
  },
  "phantom": {
    "grand_avatar": [""],
    "lurking_horror": [""],
    "spectral": ["ant", "bat", "bee", "centaur", "dragon"
       , "drake", "fish", "ghost",
      "giant", "hydra", "kraken", "lion", "lizard", "naga
         ", "snake", "spider", "thing", "worm"],
    "worldbinder": [""]
  },
  "undead": {
    "bog_body": [""],
    "death_cob": [""],
    "ghost": ["dragon", "eater", "flayed", "missing", "
       stalker", "void", "warrior"],
    "hulk_rotting": [""],
    "jiangshi": [""],
    "lich": ["ancient", "king", "lord", "mage", "
       necromancer", "summoner"],
    "macabre_mass": [""],
    "manes": [""],
    "mummy": ["dragon", "fighter", "guardian", "lord", "
       priest"],
    "necrophage": [""],
    "phantom": ["stalker", "warrior"],
    "revenant": [""],
    "rock_troll_monk_ghost": [""],
    "skeleton": ["bat", "cat", "centaur", "child", "
       dragon", "fish",
      "humanoid", "hydra", "hydra_lord", "lion", "naga",
         "snake", "ugly_thing", "warrior"],
    "skull": ["cursed", "flying"],
    "soul": ["drowned", "lost"],
    "spectre_witch": [""],
    "unborn": [""],
    "vampire": ["knight", "mage", "ressurected", "witch"]
      ,
    "wight": ["king", "stalker"],
    "wraith": ["freezing", "king", "shadow", "stalker"],
    "zombie": ["crab", "drake", "ghoul", "hound", "lizard
      ",
      "octopode", "ogre", "rat", "small", "toad", "treant
        ", "turtle"]
  }
}
```

A.3 Electronic attachments.

The following items are attached to this thesis:

1. `GPT_JRPG_BUILD.zip` archive that contains the game files as well as the folder with the Python scripts containing queries to OpenAI API and prompts. This folder can be found by following this path in the extracted archive: `...\GPT_JRPG_BUILD\GPT JRPG_Data\StreamingAssets\API`.
2. `README.md` file that serves as user documentation. It provides information about setting up the GPT JRPG game and OpenAI API application and also contains the game manual.
3. `PROGDOC.md` file containing an overview of the Unity project structure and main Unity C# scripts descriptions.
4. `images.zip` archive that contains images for `README.md` Markdown file.