Final Report

Team members	
Full Name	**GitHub Username**
Hoanh Nguyen Hosea Tong-Ho	SirBillyBobJoe
Joshua Browne	Joshuabrownenz
Aaron Worsnop	aaronworsnop

1. GPT Challenges

An important challenge we faced and overcame was an issue in which the language model would confuse its role as the Game Master, likely due to the number of messages sent previously and hallucinations. We solved this problem largely by dividing discussions into separate chat contexts, allowing us to segment conversation history to either keep track of back and forths between the player and GPT or generate one-off responses.

However, our group's largest challenges with GPT concern the model's information and context retention, as the technology is designed to emulate human-like speech, not store information and act on it. Even with its huge potential, we needed to adapt to its current shortcomings.

An important aspect of tackling these issues was writing optimised and stress-tested prompts. We used these prompts with OpenAI's 3.5 Turbo to generate content in our desired format for varying situations, allowing us to extract the info, evaluation or content in a predictable and repeatable manner. Evaluating its performance and making micro adjustments with the model's temperature and Top_P parameters allowed us to fine-tune and further improve our odds in getting a useful response.

However, even after our previous considerations, GPT would often be unable to correctly recall the hint count and would incorrectly judge whether a player's prompt is a request for a hint or simply conversation. This is where we focused on our hard-code's integration with the language model. To be more specific, one solution we implemented was a keyword pairing system, where when a keyword from both lists existed in the player's prompt, we were reasonably certain it was a hint, and would intercept the message, altering it to instruct GPT to respond accordingly. We also did the same in the opposite situation, where the player simply wishes to converse with the Game Master.

In the end, we had a very acceptable success rate in our system.

2. API Cost Considerations

Since the GPT API charges based on tokens, naturally we want to limit the number the usage per call. There are multiple ways to do this, including the trivial solution of using a hard limit wherein the generated content will be cut off. However, this is not an elegant approach and our group uses other methods as a "first line of defence".

Our most notable consideration concerning this cost is our instruction to GPT. If our prompts are very direct and we instruct GPT to be concise in conjunction with decreasing top_p, the resulting content is more sparse and efficient in token/word count. The Game Master's personality, being so short-spoken, concise and other qualities, is also a purposeful design decision that allows the player to remain fully immersed whilst our team uses less tokens. A win-win.

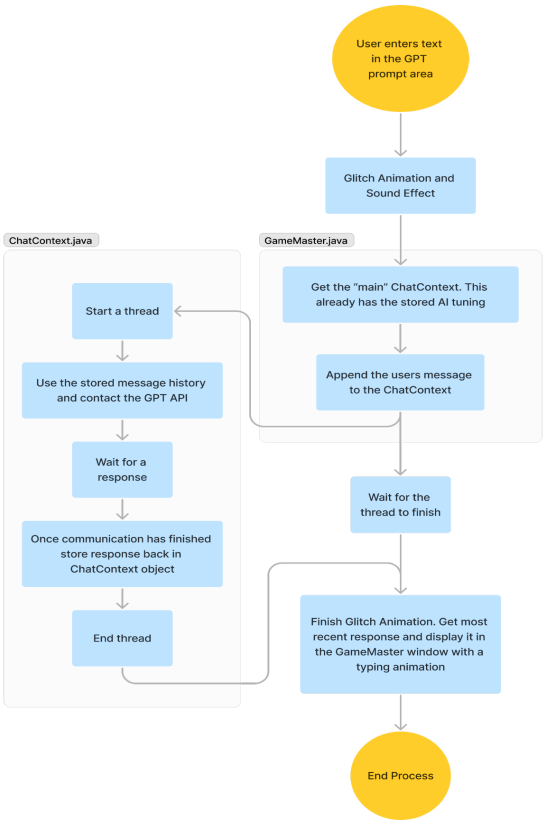
Outside of engineering GPT's responses for token efficiency, there is also reducing the number of calls made to the API. The model's generated content is extremely valuable, bringing freshness and immersion to the game, but going overboard on its implementation in your application can be very expensive. Our group found that generating an introduction and two riddles in our game is a great way to take advantage of GPT's potential, while striking a balance.

One note is that the player's direct interaction with the Game Master clearly adds greatly to the immersion of the game. Users will converse as much or as little as they wish to with our character. This uncontrollable factor in our application risks expense, but the gain in user experience makes the subsequent cost worth it.

One method of offsetting the cost of running the application at the user-end is moving the testing and development of GPT-based features to the OpenAI site. There is a free trial that exists, providing practically unlimited calls to the model that we wish to use in their playground site. Our team made use of this, allowing us to lower the costs of developing the game (especially in prompt engineering), outside of testing and user trials.

Another really important cost of GPT is the time that API calls take. This downtime is likely here to stay and can be significant in relation to modern attention spans, especially in a video game. Mitigating this, and even playing into it, becomes quite important. In our application, we played into a "calculation" or "thinking" effect for our AI Game Master with sound effects and visual effects, keeping the player immersed and engaged. This works well in the context of our Game Master, as during our concepting for it we considered this aspect of API time costs.

3. GPT API Diagram



Our GPT integration was designed in a very modular fashion. Everything was stored in the instance of a class GameMaster. From GameMaster you can create contexts which store GPT tuning values, like temperature and top_p, but also segment conversations and history. For example the main chat with the "Singularity" happened in the "main" context and riddle generation was done in the "riddle" context.

In the diagram you can see the user interaction with the singularity, this takes place in the "main" context. After this context is retrieved, we can add a message and run a specific context asynchronously and then retrieve its result once complete. In this case, the whole flow of sending a prompt to the "Singularity" takes place outside the main thread so we can do a blocking wait for the new thread to finish.

4. Prompt Engineering Challenges

GPT is not an AI Game Master in a dystopian steampunk world. To unlock the model's potential in an application like ours, GPT needs to roleplay as a character and, additionally, have resistance against manipulation from the user-hacking and even its own hallucinations. Given the nature of GPT being so reliant on input to generate responses, these issues can be solved largely through prompt engineering.

Looking more specifically into the challenges that our team faced over development, we often encountered situations where we received a response such as "Understood.", wherein GPT takes our prompt as context rather than an instruction to act on. The model would also return responses that were too excited, or not creative enough for our use-case, killing the flow of the game. The most significant issue is when GPT hallucinates, giving false information or entirely disregarding our tailored context for the game.

Our application employs text-based methods in our prompting like using ALL CAPS to signify importance, *matching the tone* of response that we are looking for and "using quotation" to separate instruction from context and conversation. This makes a huge difference in the quality of responses generated, allowing for increased accuracy, more engaging and immersive roleplay and better targeting on highly specific tasks.

The following is an example of the aforementioned prompt engineering techniques in our application. Notice how capitilisation and quotes are used. The tone also parallels our Singularity character.

> *Give the player a short concise riddle with the answer "<answer>". Begin the riddle with "I am". DO NOT INCLUDE THE ANSWER IN THE RIDDLE.*

The beauty and simplicity in all of this is how easy it really is to generate great responses by just playing into the strengths, and recognising the original purpose of the model and how it was trained. GPT is wholly concerned with text-based language, and by utilising that knowledge to your advantage, like stressing certain writing conventions, you are much more likely to be successful. Without a good understanding of this, you battle with the system and struggle needlessly.

5. Prompt Engineering Crafting

At first, it was challenging to make effective prompts as GPT has a tendency to veer on the extreme side–being too creative, or not creative at all. The model can often fail to follow instructions to say certain things, also adding in random information that is irrelevant and

unsolicited. This led to a painstaking process involving many iterations and tests before arriving at our final prompts. Even then, it still wasn't always fit for purpose.

Previously we touched on the methods that we used to get *better* responses from GPT, but at the base-level, to generate any useful content at all we must first consider the general purpose of the prompts.

With time and cost considerations, It's difficult to fit all of the instructions and content seemingly required to generate holistic responses without lengthy sentences and over-explaining. However, saying too little leads to lack-lustre results with missing information and story. Both sides of this coin have their downfalls, leading to sub-par responses and efficiency, so it was vital to be selective of where to place our focus.

Gauging the quality of different responses mainly came down to intuition, but in hindsight our metrics were whether the response fit in the tone/context of the game, whether the response was accurate and whether the response had the right amount of creativity. We found that the model is actually very successful with more simple and short prompts, rather than long ones where it often loses itself and hallucinations are introduced. After experimenting extensively with both, we found that veering on the side of simplicity was more effective, striking an optimal balance between cost and output.

Using this principle of being more short-spoken and explicit, along with our prompt engineering techniques we touched on previously, we ultimately were successful in generating accurate and fitting responses for the player to interact with.

6. Novelty and Creativity

There's a real opening to malleability in player experience with GPT's ability to adapt and recognise each new user's unique way of speaking and interacting with the application. With incorporating this form of intelligence into video games, there is the possibility of introducing increased variability, allowing no two games to be identical.

Originally, before reverting these changes to meet our clients' requirements, we implemented variable features based on difficulty factors. Hard mode had longer lengths of pipe to fix, more wires to find etc. and the shorter the time limit, the shorter mini-games and riddles were automatically adjusted to take. This more orthodox method of dynamic gameplay meshed perfectly with GPT's ability to create a custom feeling atmosphere.

However, the real possibilities emerge when using the model directly. Being able to speak to the Game Master and always receive a sensible, relevant response is huge. Gone are the days of boring NPC responses. Our group took full advantage of this, prompting GPT to roleplay immersively during riddles, giving hints and general conversation. Hints are more relevant to the player's struggles as they converse naturally and explain to the Game Master. The urge gamers have to 'try and discover every voiceline' is fulfilled to an entirely new level, with engaging and

immersive responses that just weren't possible before. The technology, and making use of the API, has really pushed our product beyond previous boundaries to immersion; removing the game-flow road-block of snapping back to reality whenever you interact with characters that was ever-present in traditional video games.

These methods of introducing variability and dynamicism play hugely into the replayability and quality of our final product–the user is much more likely to come back when value is added this way. It's clear in this one project alone how much this technology offers. We recognise that there is so much more to explore in future.

7. Software Engineering Perspective

We don't see LLMs like GPT replacing software engineers, in their current state. Instead, we recognise their use-case as incredible tools. Though we are still at such an early stage of these technologies, already by utilising tools like GitHub CoPilot, Tabnine and Chat-GPT we've seen approximately 30% improvements to our own personal productivity in both personal, university and professional programming tasks.

Currently, it is easy to offload low-value or basic, time-consuming tasks to a tool like Chat-GPT. This already frees up huge capacity and headspace for more time to spend on design or technical ideas—areas where LLMs still can't rival a skilled human developer. However, we are unsure what the future of our industry and careers will look like. On one hand, we can see how we engineers are gaining AI superpowers, who knows what the future can look like? From automated code, comments and tests to huge systems built from our ideas nearly instantaneously. This will shift our roles to be more design and architecture-focused. On the other hand, we are slightly concerned about being automated out of our jobs. It is quite easy to see how good LLMs and other tools have become in such a short space of time and we don't believe it is that far-fetched to believe LLMs (or the next generation of AI technology) could take our roles in all elements of software engineering and development.

In the short term, we're enthusiastic about the emerging tools, technologies, and toolkits that will become accessible. Yet, in the long run, the future remains uncertain. Will we retain our jobs? Will educators like you continue to work? Will the workforce as we know it even exist? Perhaps the AI revolution is just beginning…

8. Conclusion

Using LLMs like GPT is an excellent tool for creativity and incorporating new technology. It was a great learning experience. In our exploration of Large Language Models (LLMs) like GPT, we've noticed the pitfalls of over-reliance, particularly when it comes to the ideation and brainstorming phases of a project. Resorting to GPT too early in the process can inadvertently shape the direction and scope of one's work, bypassing the organic and often necessary brainstorming phase.

Over-relying on GPT might also foster a dependency where students, rather than thinking critically or researching thoroughly, may default to the convenience of LLMs. This can have implications for deeper learning and understanding. By constantly seeking immediate answers, there's a potential to miss out on the rich process of exploration and discovery, and the valuable skills that come with it.

However, on the brighter side, there's an undeniable energy and readiness among us software engineering students to embrace LLMs, stemming from a few factors:

1. **Love for technology:** Growing up in a digital age, these students are inherently comfortable with and inclined to learn about new technologies–accepting the challenges and potential, and ready to face it head on.

2. **Improved efficiency:** Without a doubt, using LLMs appropriately can improve efficiency and productivity. Students can learn more quickly, write code faster and aid themselves when completing cumbersome and tedious tasks that do little to improve their skillset.

3. **Enhanced learning:** LLMs can offer personalised feedback, assist with coding problems, or provide explanations on complex topics. This personalised touch can seriously lift traditional learning, making the entire process more engaging.

4. **Versatility:** Beyond pure software engineering, students recognise the potential of LLMs in fields like content creation, design, data analysis, and more. This versatility widens the horizons for innovative applications in their projects and future careers.

In conclusion, while it's important to recognise its limitations and shortcomings of the technology, there's no denying the huge potential that LLMs like GPT have for the gaming industry and beyond, especially from eager and aspiring software engineers.