Facilitating Game Development From Requirements to Code with LLMs

by

Ahmed El Shatshat

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Large Language Models (LLMs) are seeing further integration into the software development process, in part due to their strength as code-drivers. In defiance of prompt engineering, I put forth Software Requirements Specification (SRS) as the only necessary input to such systems. An extension of Bingyang Wei's work on a progressive prompting method to develop code, this thesis explores the capacity of Wei's Progressive Prompting Method (WPPM) to facilitate game development, using a Tailored Large-Language Model (TLLM). I investigate the following four research questions:

1. Does the WPPM facilitate game development?
2. What strengths and weaknesses emerge in the use of the WPPM to develop a game app?
3. What impacts do the results of such an exploration have on the software development process?
4. Are requirements all you need to persuade the TLLM to generate executable code that fulfills those requirements?

Through case studies exploring the main facets of game development, insights are extracted demonstrating that game development under such conditions is possible, and deserves further analysis against traditional development processes. This thesis highlights the strengths of LLMs for software development, and addresses how oversight by a domain expert can mitigate the associated weaknesses. This thesis also serves to combat dominant narratives over-estimating the capabilities of LLMs, which seek to push such systems as a silver bullet for all software problems. Further research in accurately assessing the capacity of the technology is suggested, highlighting the importance of a cost analysis against the traditional manual development process.

Acknowledgements

I would like to thank all who made this thesis possible.

Table of Contents

List of Figures

List of Abbreviations

CBS Computer-Based System vii, 2, 10, 30, 31, 38, 42

HUD Heads-Up Display vii, 24–26, 33, 34

LLM Large Language Model iii, vi, 1–3, 5–9, 28, 30–35, 37–43

PA Player Avatar vii, 10–18, 20–27, 29, 30, 32

RE Requirements Engineering 2, 6, 12

SRS Software Requirements Specification iii, 2, 3, 5, 6, 8, 12–15, 28, 33, 35, 36, 41, 42

TLLM Tailored Large-Language Model iii, vii, 3–5, 8–12, 14–38, 40–42, 47

UC Use Case 3, 8, 15, 24, 36, 41

UI User-Interface 11, 24, 33

WPPM Wei's Progressive Prompting Method iii, 3–6, 8–13, 15, 20, 25, 31–35, 41–43

Chapter 1

Introduction

1.1 Motivation

Large Language Models (LLMs) have come, disrupted most if not all industries, and now they are left in a strange place. Proponents of LLMs have continued with a fanaticism to the supposed potential of the technology, maintaining that it is a germinating silver bullet [5, 15, 12]. Others, myself included, have a more measured view, and seek to concretely assess the strengths and weaknesses of the technology in order to find where it would best fit into our software development process [14, 4, 21]. The technology itself is apparently revolutionary; as with any technological revolution, the fanfare far outstrips the subject. History can attest to such a truth [3].

There has been much ado about a process dubbed "vibe coding", in which a developer interfaces with an LLM to reach a desired vibe, or in other words a certain outcome, functionality, or aesthetic feel of a software artifact [16, 4]. Thus, the developer can approach the project from a higher level, perhaps spending more time on the creative or architectural aspects of the design. Another proposed benefit of vibe coding is the democratization of code, the ability for anyone to create a piece of software by describing the software's functionality through natural language [5]. Proponents of democratization argue that using an LLM to write code based on natural language would enable anyone who lacks knowledge in software development, but who possesses some other expertise, to create software using only the expertise E[1] possesses. For example, a pulmonary physician without knowledge

[1] "E", "em", and "er" are gender non-specific third-person-singular pronouns in subjective, objective, and possessive forms, respectively.

of software development would be able to create software that performs cardiac blood flow simulations using only an LLM and natural language.

Both concepts suffer from a fatal flaw from their outsets, which is the assumption that the LLM will perform at some minimum level of correctness and competence. This assumption already seems unlikely, as LLMs are notorious for hallucinations and are incapable of assessing context and intent as humans are [26, 28]. However, I will refrain from making any specific admonishments before describing my own experiences through this thesis.

In the advent and wake of LLMs, Requirements Engineering (RE) has largely been absent from discussions surrounding the integration of LLMs into software development workflows. This strikes me as strange, given that LLMs are, in essence, highly complex data retrieval systems whose output is largely dependent on the quality of input [7, 8]. Indeed, it can be argued that much of what constitutes prompt engineering in the context of using an LLM for code generation is merely trying to provide requirements for the desired system.

While there has been work done in the intersection of RE and LLM-assisted software development, there has not been a substantial amount of works that use real-world requirements to drive the use of an LLM. Ullrich, Koch, and Vogelsang support this claim, stating:

> "Adopting LLMs for SE depends on the compatibility of these tools with the SE process ... Good SE processes have requirements and design phases before the implementation phase ... Much of the research, however, has not considered the role of requirements and design in LLM-assisted implementation. Indeed, requirements engineering and software design are highly underrepresented in studies ... On the contrary, the studies that focus on requirements as a starting point for code generation ... do not consider actual real-world requirements (e.g., functional requirements or user stories) but rather programming tasks from coding challenges" [20].

RE as a process is about deriving what a Computer-Based System (CBS) should do, while also deriving where and when such a what may fail. When the CBS's requirements are fleshed out and well-articulated, it's designed to be as unambiguous as possible, and encapsulate the full scope of the capacity of the CBS. In other words, a Software Requirements Specification (SRS) is the perfect input to leverage the strengths of an LLM as a natural-language processor and code generation driver, the performance of which is proportional to the quality of input [13, 2]. With that in mind, my thesis includes the creation of a real-world fleshed-out SRS that covers the scope of the game app created

for this thesis. Ullrich et al.'s paper was published in 2025 during the final stages of the completion of this thesis. This thesis directly addresses the gap in research that Ullrich et al. identified. Thus, this thesis represents a novel and substantial work in this research area.

1.2 Overview and Contributions

This thesis is an extension of work completed in 2024 by Bingyang Wei, who created a method for interfacing with an LLM to consistently generate high-quality code [25]. Wei's Progressive Prompting Method (WPPM) employs a Tailored Large-Language Model (TLLM) as the core of the WPPM. The TLLM itself is created using the GPTs function provided by ChatGPT[2], and executes each step of the WPPM. The WPPM begins by inputing the SRS into the TLLM, which takes the Use Cases (UCs) provided by the SRS, and derives functional requirements for a given UC, selected by the developer. The TLLM then generates an object-oriented design that encapsulates those functional requirements, before prompting the developer to select another UC to implement. Each step of the WPPM allows the developer to review, clarify, or otherwise change the output, ensuring that the design outlined by the requirements is being met accurately.

[2] https://openai.com/index/introducing-gpts/

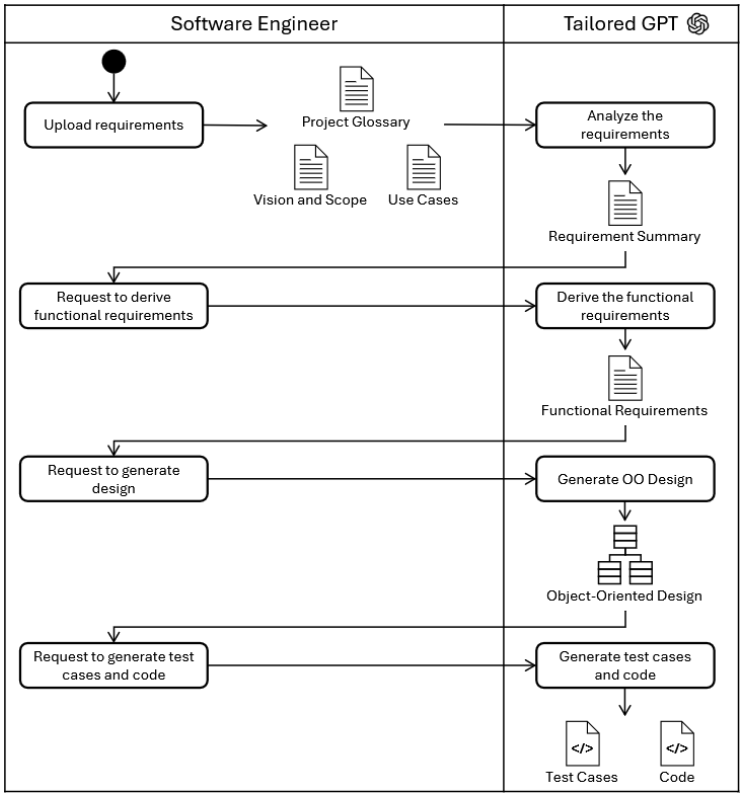


Figure 1.1: Wei's Progressive Prompting Method [25]

The foundational work by Bingyang Wei explores the capacity of the WPPM and its TLLM to facilitate the building of a Web app. Wei asked readers to apply the WPPM to other types of software projects, such as a game app, to better understand the capacity of both the WPPM and the TLLM. Therefore, I decided to apply the WPPM to the development of a game app that I had been thinking of building for some time. To do so, I made use of the TLLM built by Wei. This TLLM is built using the GPTs function provided by Open AI, which allows one to pre-prompt the GPT model[3] and provide to it a knowledge-base[4]; Wei provides these resources in his paper. As this thesis is an exploratory report focused on eliciting the strengths and weakness of the WPPM and its underlying TLLM, I wanted to preserve as much of the original WPPM as possible. As such, I did not augment or alter the TLLM configuration documents for this thesis.

This thesis is a report of my experience in applying the WPPM to the development of

[3] https://github.com/Washingtonwei/software-engineer-gpt
[4] https://github.com/Washingtonwei/software-engineer-gpt/blob/main/knowledge.md

a game app over four months, with a development time of approximately 350 hours. These hours include the time to implement the code of the game, the creation and integration of visual assets, and the testing and verification of the proper functioning of all code artifacts. The creation of the SRS is not included in this time estimation, as those documents were steadily built up during the course of my two-year degree; the bulk of the work on the SRS was done closer to when development on the game app began. The game app is built using the Godot game engine, and its source code is 3852 lines; while this may seem a bit small, it is important to remember that the Godot game engine itself handles a sizable amount of game logic. The contributions of this thesis are that it

- demonstrates the extent to which a TLLM can facilitate game development using the Godot game engine through the WPPM outlined in the seminal work by Bingyang Wei,
- assesses the strengths and weaknesses of LLMs and critically reflects upon the rhetoric surrounding it, and
- provides, in an online appendix, all the artifacts that I, the TLLM, and Godot produced during the experience for examination by the reader and for use in future research[5]

1.3 Related Work

Wang et al. compare various Natural-Language-to-Code generation models, both Code Search and Code Generation types, and test their effectiveness for transforming natural language descriptions into source code [24]. Furthermore, the authors compare pre-trained models against models without pre-training under both types, and even go so far as combining different techniques into one synthesized output. Their findings support the idea that the application of such techniques lead to a high degree of quality code output. ChatGPT is cited by Wang et al. as being weaker for code generation than the techniques used in their study. However, their findings were published in 2023, and LLM technologies have been extended and improved since publication, as code generation driven by LLMs increases in popularity.

Mitchell and Shaaban present a theoretical framework for correcting the errors associated with vibe coding [15]. Such errors include constraint inconsistency, in which the

[5] https://github.com/aelshatshat/Ahmed-E-Master-s-Thesis-My-Adventure

addition of new functionality contradicts the behaviour of implemented code, or constraint-reconciliation decay, wherein the LLM is focused on pattern completion more than fulfilling the vision across prompts, leading to decaying rates of success as project scope increases. Their proposed framework brings in vibe reasoning using formal verification, also driven by LLMs. Such an approach runs in reverse of the WPPM, Mitchell and Shaaban instead generating SRS using the LLM through natural language input, with the generated SRS serving as the verification metric. Despite how such a method runs contrary to the WPPM, the authors reinforce that human oversight and verification is essential to maintaining quality of output, which aligns with my view.

Vaithilingam, Zhang, and Glassman conducted a user study to better understand how programmers use and perceive Github Copilot [21]. They selected three Python data processing tasks of varying difficulty, and tested the capacity of their participants to complete the tasks; once using Intellisense, and once using Copilot. While the majority of participants stated they preferred to use Copilot, the results showed little improvement between the methods. This lack of improvement is attributed to the code generated by Copilot often containing bugs, with some participants stating further that they were unable to understand the output from Copilot in the first place. These are issues that RE is designed to address, and further reinforces the importance of a domain expert in interfacing with an LLM. However, it could also indicate that these LLMs introduce bugs that a human would not be able to easily find the source of.

From a measured theoretical perspective, Maes provides a general review of AI-based coding, being a pioneer in the space [14]. He presents the many challenges associated with a pure vibe coding approach: architectural inconsistency and technical debt acceleration, issues with debugging and troubleshooting due to unfamiliarity with the code or needing to rely on the LLM to perform debugging, as well as documentation and knowledge transfer problems due to code being written only through prompting, leading to a loss of intent and context. To address these problems, he presents a framework for AI-driven coding based around the verification of the integrity of AI-generated output and ensuring all components are understood and maintain functionality.

Ullrich, Koch, and Vogelsang interviewed 18 software practitioners from 14 companies across 12 domains, to understand how these practitioners incorporate requirements and design artifacts in LLM-assisted implementation of code [20]. They found that developers break down requirements into programming tasks. Doing so reportedly promotes better quality LLM output than does inputting requirements as-is. They found also that the methods in which the practitioners interacted with an LLM could be separated into three categories:

1. manual coding with intelligent auto-complete,
2. incremental code generation, in which the developer uses generated code, but manually tweaks the output, sometimes reformulating the prompt for better output, and
3. extensive code generation, in which the developer tries to achieve all requirements using generated code, iterating using an LLM until the output achieves the requirements.

The researchers conclude that, while LLMs are actively used in practice, their use is preceded by manual and creative steps that require domain expertise and expertise in requirements and software engineering.

Haque reviews and critically analyzes the impact of LLMs on the software development process, and highlights key strengths and weaknesses of various LLMs [8]. While he notes the successes of LLMs in generating scaffolding code and identifying potential bugs, Haque explains that LLMs lack the deep contextual understanding required to identify more esoteric issues dependent on specific contextual knowledge and analysis. Furthermore, LLMs produce code that is syntactically correct, but doesn't function in practice. Haque states that LLMs can be use in small-scale refactoring, but major architectural refactoring requires human expertise and oversight, as LLMs struggle to keep a holistic and semantic understanding of a large codebase in context. He observes that LLMs are unable to properly handle novel or rare problems, and lack transparency in output. Haque concludes that while LLMs are powerful tools, they should serve to only augment human capabilities and not replace them, especially since human oversight is necessary for the effective use of an LLM.

While there has been some work done in the space of LLM-assisted game development, to my knowledge, none of it goes beyond generating simplistic or prototypical examples of game apps. Grow and Khosmood interview participants of a ChatGPT Game Jam, where participants were asked to code a game app through use of ChatGPT [6]. Participants were told to spend no more than 6 hours on the game itself. This instruction necessarily limits the insights that can be gathered from the study. Vivas built a game prototype with LLM assistance [22]. In addition, he compared the quality of outputs from various LLMs in fulfilling game requirements, which were formatted as prompts. However, the game app does not go beyond the implementation of prototypical features and assets. Furthermore, requirements for this project do not appear to have been completed before coding ensued. Ternar et al. perform a qualitative review of the literature on generative AI in game development [19]. They highlight that efficiency is not a guaranteed property in the use of generative AI, and that its use is dependent on phase and context. Ternar et al. also report that human intervention is necessary for proper integration of LLM output.

The work most related to mine was done by Wei[6], who provides a new method for interfacing with LLMs, the WPPM [25]. This is a method that begins with the completion of the SRS, which contain a list of UCs. The developer feeds the SRS into the TLLM input, after which point the TLLM prompts the developer for a UC to implement. After the developer chooses a UC, the TLLM generates functional requirements that achieve the chosen UC. Between each step of the WPPM, the developer is expected to review the output, and alter any components that do not align with the vision outlined in the SRS. Afterwards, the TLLM generates an object-oriented design fulfilling the functional requirements; upon the developer's confirmation of that output, the developer can generate test cases to validate the functionality. Under the WPPM, the benefits of using an LLM as a code generation driver can be achieved, while avoiding many of the pitfalls; the use of a well-articulated SRS as input and the developers serving to verify output across multiple stages keeps code high-quality and within scope. The SRS must be done before any coding ensued, so that the developer has a clear vision on what needs to be completed and can accomplish that goal with limited confusion.

In the rest of the thesis, Chapter 2 covers the game development process through case studies highlighting different parts of the game app. Chapter 3 contains a discussion of the results of the game development process, and situates the results in a broader context. Chapter 4 concludes the thesis, discussing the limitations of the thesis and future work, before providing a final conclusion to the findings of this thesis.

[6]Of course, this is no accident since my work is an extension of his work!

Chapter 2

Research Questions and Method

To guide my research and highlight the main facets of this thesis, I formed the following research questions:

- RQ1: Does the WPPM facilitate game development?
- RQ2: What strengths and weaknesses emerge in the use of the WPPM to develop a game app?
- RQ3: What impacts do the results of such an exploration have on the software development process?
- RQ4: Are requirements all you need to persuade the TLLM to generate executable code that fulfills those requirements?

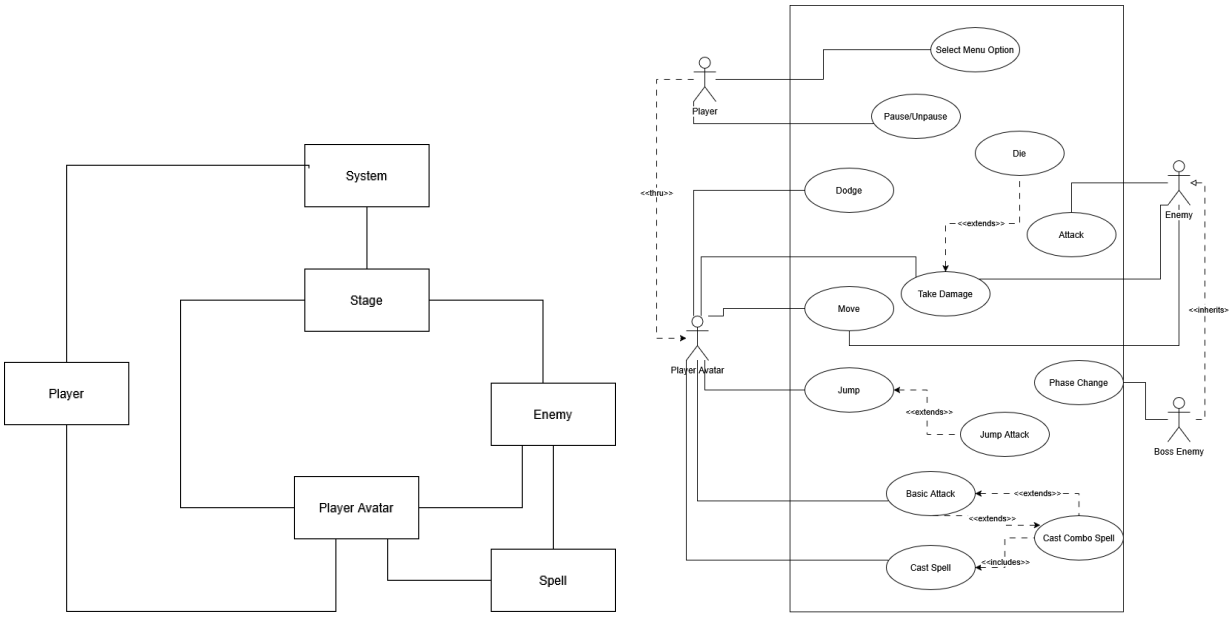
Wei's seminal work demonstrated the capacity of the WPPM to facilitate the development of a medium-sized Web application. While impressive, it is important to gather results across the domains of software development, to better understand the capacity of LLMs as a core component of this method.

To that end, this thesis centres around the development of a complex 2-dimentional action-adventure game app. Conceptually, it's important that the gameplay constructs and their associated systems be novel within the space. Otherwise, there's the possibility that the TLLM would simply retrieve large chunks of data that already exist within its dataset. It's impossible to have every single aspect of the game app be novel; what's important is that it possesses its own unique identity as a game. As an example, it would be banal to

create a Snake game app or otherwise extend its functionality, given how common such projects are. In short, the structure of the game app used in the research and its available systems must be sufficiently novel and complex to give the WPPM and its TLLM a decent test.

2.1 Game App Design

The game app, titled *My Adventure*, is a 2-dimensional side-scrolling action-adventure game. The player takes control of Ahmed, who is the Player Avatar (PA), on a quest to defeat powerful foes. Gameplay is structured as a series of Boss Enemy encounters, separated by unique arenas across a world map. The game app takes inspiration from *Kirby Super Star*[1], as well as from my previous game app project *Kill the Lich*[2].



(a) Domain Model (b) Use Case Model

Figure 2.1: The Domain Model and Use Case Model of the CBS

[1] https://en.wikipedia.org/wiki/Kirby_Super_Star
[2] https://godofdragons.itch.io/kill-the-lich

To demonstrate the capacity of the WPPM and its underlying TLLM for game design, it was important to exercise their capacity across the three main code artifact categories that manifest in game development. These are:

1. Player-controlled artifacts, typically the PA

Figure 2.2: The Player Avatar

2. System-controlled artifacts, such as enemies and attack/spell effects

Figure 2.3: Examples of system-controlled artifacts

3. Top-level artifacts that lead the player to and inform them of the status of their gameplay, such as menu options and User-Interface (UI) elements

Figure 2.4: Examples of top-level artifacts

There was no need to complete an entire game; I argue that it was sufficient to implement and flesh out only the most complex of these artifacts as would manifest in game development. The underlying logic for these artifacts and their associated systems are largely transitive, and a less complex artifact would effectively be a subset of a more complex artifact. While it is theoretically possible that unique issues could manifest in the development of an artifact that would invalidate the benefits of the WPPM, identifying such cases before encountering them is bordering on the impossible. This is somewhat problematic, as the point of RE is to give a complete SRS. However, I argue further that the developed game app is effectively a complete demo with a playable executable, and thus the SRS is complete under that scope.

Therefore, for the player-controlled artifact, I implemented one PA that provides a range of gameplay options to the player. These gameplay options included a basic attack-combo system, a spell-casting system with multiple spell types, an air-attack system, and a dodge function. For the system-controlled artifact, I implemented one Boss Enemy with multiple diverse attacks at its disposal; the effects of PA spells also fell under system-controlled artifacts. Finally, I implemented the necessary top-level artifacts to both get the player to gameplay, and inform them on their status during gameplay.

These systems were conceptualized and iterated upon on the Notion Web-app[3], using a game design SRS template[4]. Please note that, while Notion possesses AI functionality, the AI functionality was not used for this project. The SRS document created using Notion's platform served as input to the TLLM to facilitate development of the game app. I created

[3] https://www.notion.com/

[4] https://www.notion.com/templates/game-design-document

visual assets using Adobe Photoshop[5] and Aseprite[6], and developed the game app using Godot game engine version 4.4.1[7]. It's worth noting that I had no development experience using Godot prior to beginning this project, and learned much of how to use Godot as a game engine through the WPPM itself. The game app executable is not hosted on the linked GitHub repository due to size limits, but it can be created using Godot and the project files.

It is difficult to describe the whole of the game development process in rhetoric. Instead, the game development process is separated into case study categories based on the three code artifact categories. The development process did not involve creating all artifacts of each type linearly. So, these case studies should coherently frame the development process and provide a basis for discussion.

2.1.1 Case Study 1: Player-Controlled Artifact

To start, I implemented the player-controlled artifact, the PA. In my SRS, I derived a set of requirements describing the possible actions a player could take through the PA. The primary actions a player can take are:

- The PA moves left and right with the A and D keys respectively.
- The PA jumps using the SPACEBAR key, pressing the SPACEBAR key again when in mid-air for a double jump.
- The PA attacks with a sword when the Left-Mouse Button (LMB) is pressed. Pressing the LMB while the previous attack's animation hasn't finished extends the attack into another step of the attack-combo. There are four steps in the attack-combo, which can be buffered by pressing LMB again during the current combo step. The attack-combo resets if it finishes or if the player does not provide input.
- The PA casts a spell using the Right-Mouse Buttom (RMB). There are three types of spells: Instant, Incantation, and Combo. Instant Spells cast instantly, Incantation Spells have a delay during which the PA cannot act before they are cast, and Combo Spells take the place of a step in the attack-combo.

[5] https://www.adobe.com/ca/products/photoshop.html
[6] https://www.aseprite.org/
[7] https://godotengine.org/download/archive/4.4.1-stable/

- The PA performs an air-attack when pressing LMB in the air, during which the PA spins and does damage in an area.
- The PA performs a Dodge Spell when the SHIFT key is pressed; the standard Dodge Spell moves the PA quickly in the facing direction, and provides some milliseconds of invulnerability. There is a short delay before this can be used again.
- The PA loses a resource called Health when attacked by an enemy. The PA dies when it loses all its Health, forcing the game to restart.
- Spell-casting takes a resource called Mana, which recovers slowly over time or when an enemy is hit with the basic attack, at a fixed amount.

Note that these listed requirements are a summary of the implemented PA requirements, and are reflected verbatim neither in the SRS, nor in the input to the TLLM. The TLLM retrieved the requirements directly from the SRS's use case document, which is available in the Appendix.

Control of the PA, which constitutes the core of the primary gameplay loop, was highly dependent on feedback from gameplay testing. However, there first needed to be some base PA functionality in place before this functionality could be tweaked based on the results of PA gameplay testing. As such, this case study is separated into a first pass that covers the initial implementation of the PA requirements, and a second pass that covers the work that was done after testing PA gameplay with other implemented systems.

First Pass

The PA is the core of the game; without a way for a player to interact with the game world, one can't really call the app a game at all. As such, the first pass was focused on getting the requirements and associated systems in place. Godot uses a Node-based architecture: all code artifacts are built on top of the engine's base type, called a Node. The Node class has derived subtypes that Godot provides, with these subtypes providing additional baked in functionality. Sub-Nodes can be attached to a parent Node, allowing for the decoupling of code artifacts.

I started with implementing the PA's movement. The TLLM generated a basid Node hierarchy for the PA, as well as boiler-plate code for the movement. I attached the relevant sprites to and animating those sprites through the PA's AnimatedSprite2D Node. Afterwards, I implemented the PA's ability to jump with code generated by the TLLM. Once

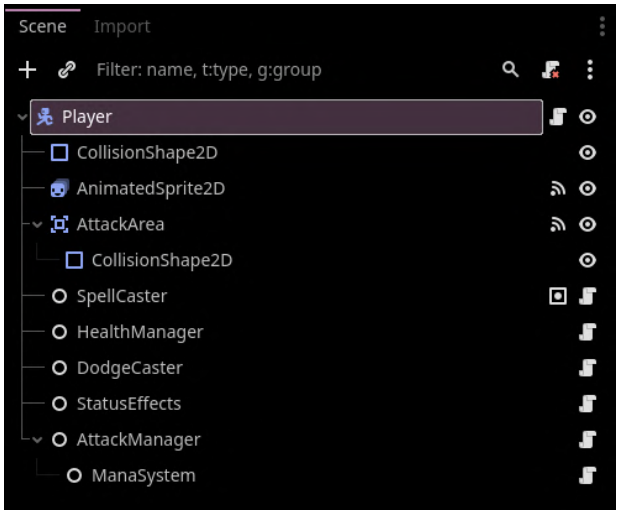


Figure 2.5: The Node structure of the PA

movement was in place, I added collision detection to the PA so it stays within the game area's bounds.

At this point, I reviewed the UCs and noticed that the TLLM added UCs that weren't defined in the input documents. I was able to alter the listed UCs through prompting the TLLM directly. I decided that it would be safer to add or otherwise change any given UC by prompting the TLLM directly, rather than by uploading an updated SRS. I would instead update the SRS documents myself in parallel to avoid architectural drift. The rest of the steps of the WPPM proceeded as normal, following these UC injections.

When it came to implementing the attack-combo system, the TLLM was capable of retrieving various designs that satisfied the requirement. This included using a timer to force a delay in-between attacks, or using an AnimationPlayer Node to play animations and set the value of variables through the animation timelines within the AnimationPlayer. However, the design I ended up going with was a system using the current frame of the currently playing animation to drive the associated combo logic: whenever a frame change was detected, a function called `on_animation_frame_changed()` is called. During the execution of said function, variables could be set and functions could be called at specific frame indices of any given animation. This allowed for more flexibility and precision when designing gameplay options for the player, such as allowing the PA to exit a combo on a frame earlier than the attack would typically finish if the player wanted to move instead of continuing their attack-combo.

Afterwards, I prompted the TLLM to implement the air-attack UC, in which the PA

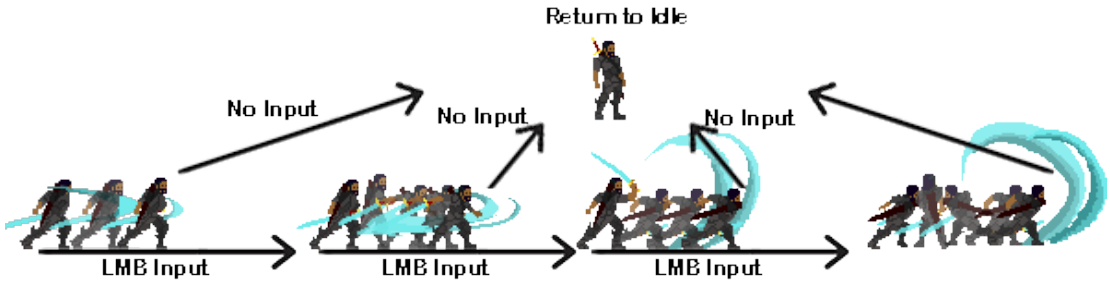


Figure 2.6: PA Basic Attack-Combo

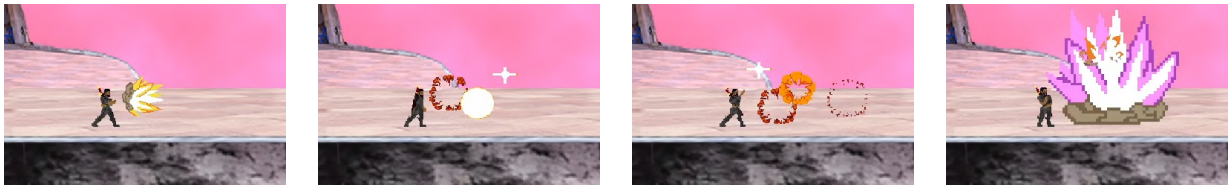
spins in the area, dealing damage in a circular area. The placement of the sprites on the spritesheet caused some issues visually, as the air-attack sprites were placed too much higher than the other sprites. The TLLM recommended changing the animation offset, but this caused all animations to be offset. In the end, I went back and created a new spritesheet for just the air-attack sprites, thereby solving the problem.

Implementing the spell-casting system required the implementation of a general Spell Resource object, which could then be extended to the three specific Spell types: Instant Spells, Incantation Spells and Combo Spells. The TLLM recommended the creation of a SpellCaster Node to serve as a decoupled artifact in control of the spell-casting logic, and responsible for storing the currently equipped Spell resource. This SpellCaster Node would be supported by a ManaSystem Node, responsible for managing the Mana resource.

Once these foundational systems and abstract types were in place, I created a Fireball Instant Spell through prompting the TLLM. While I consider the spell effects themselves to be System-controlled artifacts, I believe it makes more sense to include them with the Player-controlled artifacts for the sake of discussion. The Fireball's behaviour is quite simple: it moves horizontally from the point of firing and changing to an impact animation effect on collision with an enemy. Thus, the Fireball Spell took little time to implement. I continued by prompting the TLLM to implement an Incantation Spell; this spell type required an additional check and associated logic for the casting time. However, I didn't go beyond using test assets and spell behaviour for this initial Incantation Spell implementation.

I went on to implement the Combo Spell, which is a four-step combo attack using explosions, eponymously named "Magic Explosion". These combo steps allow for the player to weave both basic attacks and magical attacks together, using the PA's unified attack-combo structure. The TLLM struggled to fulfill this requirement despite my reformulation of the prompt and my attempts to progressively coax the TLLM towards the proper functionality.

In the end, I decided to go fully hands-on and corrected and debugged the functionality manually. However, I still made use of the code artifacts the TLLM generated, as well as the general design structure of using a decoupled script to handle the spawning of explosions at each combo step. I felt this method of implementation would take less time than starting the Combo Spell implementation from scratch.



(a) Magic Explosion 1 (b) Magic Explosion 2 (c) Magic Explosion 3 (d) Magic Explosion 4

Figure 2.7: The Magic Explosion Combo Spell

I then prompted the TLLM to implement the Health resource, as well as the requirement of taking damage and getting to a death state. This was done with few issues, aside from those relating to the UI, whose issues are discussed in the description of the third case study.

Following that, I prompted the TLLM to implement the Dodge Spell requirement. Initially, the TLLM folded the functionality of the Dodge Spell into the spell-casting functionality. I did not like this structure, so I clarified my prompt and told the TLLM to create a DodgeCaster Node to handle the Dodge logic, in a manner similar to the SpellCaster Node. The DodgeCaster is responsible for handling the cooldown of the equipped Dodge Spell, as well as applying the appropriate duration of invulnerability to the PA when a Dodge Spell is used. I also added the double-jump functionality myself at this stage, as having a single jump felt restrictive to gameplay. Finally, a test enemy was created by prompting the TLLM, and was used to test hitbox detection and damage calculation for the PA's attacks.

Second Pass

With core systems in place and a test environment having been set up, the PA's behaviour could be modified based on the results of testing in that environment. These modifications to PA behaviour ranged from more tweaks, such as allowing the player to turn the PA's facing direction in mid-air using the Dodge, or allowing the PA to attack and cast spells normally in mid-air if they do so while falling. Much of the Boss Enemy's functions had

been implemented by the time of this second pass, so this pass also involved connecting some of the hooks exposed by the Boss Enemy.

I designed a Boss Enemy attack to add a slow effect to the PA if it connected. Thus, I prompted the TLLM to implement a status effect system to handle these types of effects. It responded with the implementation of a StatusEffect Node, which is attached to the PA, and manages any status effects applied to the PA. Some other Boss Enemy functions necessitated a rework of the PA's physics, which was done along with the addition of a knockback function, allowing for enemy attacks to push the PA back in the direction of the attack. The TLLM gave an initial implementation for these reworked physics, but I wasn't happy with how it worked. I ended up altering the implementation to better suit my vision, but still made use of some of the TLLM's generated code.

For the attack-combo, the first attack by the PA was not being registered as a hit. I deduced that there was likely a race condition in processing the animation frames, and certain frames were being skipped because the CPU was elsewhere in the code. Before diagnosing the the source of the bug myself, I did ask the TLLM if it could tell me where the bug was coming from, but it could not.

To solve the race condition bug, I decided to refactor the basic attack-combo system to be set up in a manner similar to the spell-casting system, with a central AttackManager Node responsible for processing the attack-combo system logic; this is conceptually parallel to the SpellCaster Node in the spell-casting system. Classically, a refactor would require identifying, extracting, and re-implementing the attack logic in such a new Node. From experience, I had originally planned one to two days for the refactoring. However, the TLLM was able to process both the refactored design, written in natural language, as well as the implemented code, and generate code reflecting the refactored design within an afternoon. With the updated design, the bug was fixed and no further issues manifested, and the design benefited overall from the decoupling of systems.

Through my testing, I felt the PA's having access to only one spell was restrictive to gameplay. Thus, I looked to extend the spell-casting system to allow the player to swap between three spells using the 1, 2, and 3 number keys respectively. While this isn't a very complex change, it is a tedious one; it would also require the implementation of visual feedback to show the changing of spells. After inputting the updated spell-casting requirements to the TLLM, I completed the extended functionality within two message turns; I integrated the associated visual artifacts not soon after with no issues.

Finally, I prompted the TLLM to implement a proper Incantation Spell: the Magic Missilerack. This spell creates a magical missilerack that appears behind the PA, and shoots out a volley of enemy-tracking magic missiles once the cast time has elapsed. In

addition, I prompted the TLLM to add a progress bar that appears when casting an Incantation Spell to visually indicate the elapsed cast time. I added also a magic circle visual effect to the casting animation, and used the same magic circle effect to provide more visual feedback to the double-jump.

(a) Casting Step (b) Cast Complete

Figure 2.8: The Magic Missilerack Incantation Spell

Figure 2.9: The Celestial Primate

2.1.2 Case Study 2: System-Controlled Artifacts

The focus of Case Study 2 was on the implementation of the Boss Enemy, known as the Celestial Primate, the creation of the Celestial Primate's attacks, the implementation of a CutsceneController artifact for in-game cutscenes, and the implementation of a tutorial system. PA Spell effects are discussed as part of the first case study, and so do not appear in this case study.

Implementing the Boss Enemy

As the goal of this thesis is to test the limits of the capacity of the WPPM in the domain of game development, there's little reason to implement a simplistic enemy type. What should be tested is if the WPPM has the capacity to facilitate the development of a complex, multi-phased entity; in other words, a Boss Enemy. It can then be reasoned that, if a Boss Enemy can be developed, then more simple enemies can be developed as well.

I had already created a base Enemy class for the sake of testing some of the PA's functionality by prompting the TLLM. This Enemy class includes the ability to take damage and die, so these did not need to be re-implemented. I prompted the TLLM to extend this Enemy class to an abstract Boss class, which the Celestial Primate's script extends; this abstract Boss class includes the phase change logic. I was unsure how the Boss Enemy's architecture should be set up. So the initial stages of the implementation involved some exploration and learning. After I understood how the Boss Enemy's architecture should be structured, I began working on the Celestial Primate's attacks.

Creating Boss Enemy Attacks

The Boss Enemy was designed to have 4 primary attacks, plus a special attack that occurs between boss phases; this special attack was called the mid-combat event. The Boss Enemy's attacks are:

- The Celestial Primate shoots a beam of lightning at the PA's current position.
- The Celestial Primate creates a mist of cold air at the PA's current position that solidifies into ice after a period, dealing damage in an area.
- The Celestial Primate shoots a swarm of meteors that track the PA's position for a short period.
- The Celestial Primate creates pillars of flame that erupt from the floor, evenly spaced across the level.
- The Celestial Primate creates a black hole in the center of the level, the Chaos Zone, that pulls the PA into it, dealing damage if the PA is caught in its event horizon. This is a mid-combat event, that occurs between Phase 2 and Phase 3 of the Boss Enemy.

This list is effectively a summary of the Boss Enemy's attacks' requirements; the Use Case Document can be found in the appendix, and the full SRS has been provided in the online repository.

The first attack I implemented by prompting the TLLM was the Lightning Bolt. I was still focused on implementing the base functionality and understanding how the different components of the artifact should interact, so I wasn't too worried if the Lightning Bolt attack didn't fully match my vision. I could, and did, go back to alter the functionality once I was more familiar with the implementation process.

Before attaching any attack to the Celestial Primate, I prompted the TLLM to create a test scene with a test Node that used the attack. I verified the Lightning Bolt's functionality to be satisfactory for the moment, and moved onto implementing the Meteor Swarm attack. The TLLM misinterpreted some nuances of how I wanted the Meteors to behave, but I was able to reformulate the prompt in a way that generated an output I was happy with. My implementations of the Flame Column and the Ice Shock attacks by prompting the TLLM came after, with few issues manifesting in their implementations. The Ice Shock attack included the application of a slow effect to the PA if it damaged the PA, which is

mentioned in the first case study. After I felt satisfied with the function of all these attacks, I prompted the TLLM to fold all the attacks into the Celestial Primate's behaviour.

Finally, I implemented the Chaos Zone Mid-Combat Event. Originally, I imagined various zones appearing throughout the arena, but I decided to simplify the behaviour to just one zone appearing in the center of the arena. The suction effect of the Chaos Zone required the restructuring of how the PA processed its physics.

I wanted Mid-Combat Events to have multiple paths to success. Therefore, I added specific functionality that provided to the PA immunity from the Chaos Zone. This takes the form of the appearance of a lotus seat beneath the Chaos Zone, before the Chaos Zone's effect begins. If the PA is motionless over the lotus seat when the Chaos Zone effect begins, a special animation of the PA sitting on the lotus seat plays. Text appears on the screen indicating that the player has engaged the Yin-Yang Defence, providing visual feedback and indicating success. If the player makes the PA move too early, these visual effects change to indicate failure, and the PA is sucked into the Chaos Zone.

(a) Appearance of the Lotus Seat

(b) Successful Attempt at Yin-Yang Defence

(c) Failed Attempt at Yin-Yang Defence

Figure 2.10: The Chaos Zone Mid-Combat Event

The TLLM was able to generate code for the disparate elements of the Chaos Zone Mid-Combat Event, but was unable to connect these elements together functionally. I took these generated elements and fixed the bugs that were manifesting from their integration, with the final product working as expected.

With a better understanding of how to implement Boss Enemy attacks, I went back and fixed the Lightning Bolt attack to be more in line with my original vision, again by

prompting the TLLM. There was some struggle with getting the TLLM to generate an output that fully captured my vision, but the attack was finally implemented by taking the almost-correct code generated by the TLLM and altering the parts I wasn't happy with.

After doing so, I implemented the Celestial Primate's movement by prompting the TLLM. I decided to have the Celestial Primate's movement be based on anchor points placed throughout the arena. An anchor point stores the possible movements the Celestial Primate can make from its current anchor point. I attached a MovementManager Node to the Celestial Primate to handle the storing of anchor point references, and the execution of movement. The TLLM was told to generate code for this design, which it provided.

Three MovementPattern scripts were created by prompting the TLLM: movement in a straight line, movement in an arc, and a teleport. The MovementManager stores these MovementPatterns, and executes them when the Celestial Primate moves. Finally, I did some minor tweaks to the Celestial Primate's behaviour, both manually and by prompting the TLLM. This included improving how the Celestial Primate decides its next action, as well as enhancing visual effects and animations, such as adding a visual effect when a phase change occurs.

Integration of the Cutscene Controller

The implementation of gameplay cutscenes required integrating functional components across many different code artifacts, primarily the PA and the Boss Enemy; they would serve as the main actors of any given cutscene. Implementing the starting cutscene through prompting the TLLM took little effort given the simplicity of the interaction: the playing of two animations and delaying player control and Boss Enemy action until the cutscene was finished.

Implementing the ending cutscene took significantly more effort than the opening cutscene, as the game state prior to the end of the fight is much more variable, in contrast to the fight's beginning. Furthermore, this ending cutscene had three parts to it. The first was in the game's normal style, allowing the Celestial Primate's death animation to play and then having the PA approach the defeated Primate. The second part was in a slideshow style, with panels I created to deliver a short exchange between the Primate and the PA. Finally, the game returns to the standard art style and both actors perform animations exiting from the level.

Given this increased level of complexity, both in resolving game state and progressing through these various stages of the ending cutscene, the TLLM struggled to generate code

Figure 2.11: Celestial Primate Ending Cutscene

that fully achieved this. However, it laid enough groundwork and gave enough context such that I was able to understand how the different components should interact and debug it myself from that point on.

Implementation of the Tutorial System

After the Celestial Primate Boss Enemy encounter was finalized, I decided to integrate a tutorial section the plays prior to the beginning of the fight. This would teach a new player how to control the PA, and give them some hands-on testing before the fight proper began.

I created some textual visual assets that progressively explain the different mechanics available to the player through the PA. Then, I prompted the TLLM to fulfill the new tutorial UC. It created a TutorialManager Node, with a script that checks for certain signals and progresses through the different visual elements as conditions are met. This implementation was straightforward, so no major issues manifested. I also added a special condition that skips the tutorial if the player attacks the Celestial Primate before the tutorial is finished.

2.1.3 Case Study 3: Top-Level Artifacts

This case study covers the implementation of the Heads-Up Display (HUD), which is a UI artifact that displays the PA's Health and Mana resource bars, and its three equipped spells. This case study also covers the implementation and refactoring of top-level system artifacts.

Figure 2.12: The First Tutorial Message

Implementing the HUD

When I began to implement the Health and Mana resources for the PA, I used temporary resource bar assets. The TLLM struggled to provide good guidance on how to implement even these temporary assets, and so I ended up implementing them manually. Later, I created proper visual assets for the HUD, and attempted to use the WPPM to integrate these elements. I knew from the implementation of the temporary assets that the TLLM was not going to be very useful, and so I integrated the proper HUD assets manually. The HUD also had some code elements that the TLLM was able to generate with few issues. I also added a Health bar for the Celestial Primate.

Figure 2.13: The PA's HUD

Implementing and Refactoring Top-Level Systems

I created top-level artifacts Game, GameWorld, and GameState at the beginning of the development process by prompting the TLLM, as part of the initial scaffolding process.

After the game app reached the final stages of development, I added a pause function that allows the player to resume, quit to the map, and quit the game by prompting the TLLM. I also prompted the TLLM to add a game over screen that appears when the PA dies, which allows the player to restart, quit to the map, or quit the game. I added the map screen, and implemented a projection visual effect that plays when the player selects a level, which provides more information on the selected level. In contrast to the TLLM's struggle with the HUD, the recommendations for how to structure this visual effect were relevant enough to be useful in the implementation. The bulk of the implementation of this visual effect was still done manually.

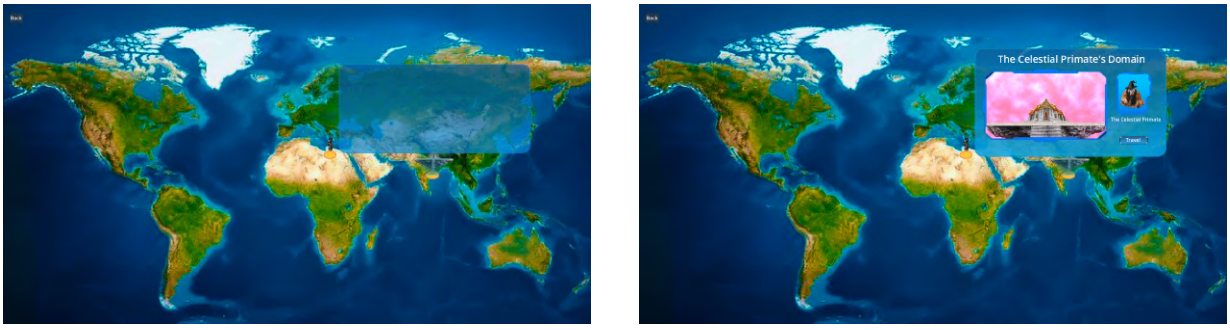


Figure 2.14: The Map Screen and Level Select Projection

When it came time for me to connect various scenes together and establish a gameplay flow, that is, to allow the player to get from title screen to gameplay and back, I found that there was a fundamental problem with how the scene hierarchy was set up. Erroneously, I

had been adding the PA and game level Node artifacts directly to the abstract GameWorld Node, which should instead be loading PA and game level scenes dynamically as sub-Nodes. After describing the issue in natural language, the TLLM generated a refactored design which, while not perfect, could then interpolated through my own refactoring process. The TLLM was also referenced for clarifications and sanity checks as the refactoring progressed, with the final result working as expected. Finally, I added a fade effect that plays when the game screen changes. This was done with help from the TLLM, but I still needed to do some extra semantic corrections to ensure proper functionality.

Chapter 3

Results

3.1 Case Study 1: Player-Controlled Artifacts

For boiler-plate code implementation, such as basic movement and attacking, the TLLM was very proficient. This isn't necessarily surprising given how LLMs function; however, the TLLM's ability to lay a decent foundation that can then be tweaked by the developer to accomplish a specific application is a theme in this paper.

Something that I did find surprising, however, was how adept the TLLM was at providing game engine-specific information to guide the implementation. Within the TLLM's dataset appears to be a wealth of information about Godot and how to, for example, set up the engine's base types, called Nodes, within any given scene and what parameters to check and change. It wasn't always perfectly correct; output from the TLLM sometimes referenced to Node parameters that didn't exist in the version of Godot being used, or generated code that missed some functional requirement when tested, or otherwise caused an error when ran. These issues were typically found and addressed soon after, sometimes with additional prompting to the TLLM. The TLLM also served to direct me towards certain Node types within the game engine that I wasn't aware of, which helped me to parse the numerous Node types that Godot provides.

Unfortunately, there was a sense of volatility attached to interacting with the TLLM. I found myself policing the ways in which I should interact with the TLLM in fears of breaking the interaction in some way. For example, I was afraid to input updated SRS documents in fears of corrupting the TLLM's context, and so decided to feed in additional requirements directly as prompts.

Implementing the Instant Spell type was quite simple, as it required only the spawning of a spell effect on RMB press. Incantation Spells were similarly simple, as it required checking that the type of the spell was an Incantation, and delaying action for the period of time referred to as the cast time, conveyed to the player through the PA performing a casting animation with a progress bar appearing above the PA, before spawning the spell effect. The TLLM facilitated the implementation of these artifacts with little to no problems. As an addendum, given my lack of experience with Godot, the way in which certain game engine constructs, such as Resource-type objects, functioned in the engine caused me some confusion. To this end, the TLLM was quite useful for clarifying the intent behind and the usage of such objects when prompted for such information.

Combo Spells were more complex to implement given their interaction with the basic attack-combo system. While the TLLM was able to generate code that seemed to implement the Combo Spell logic, upon further testing there manifested errors and other bugs due to the interactions between the PA, whose script at this point contained the attacking logic, and the SpellCaster, which was in charge of the spell-casting logic. Here, my domain knowledge was essential to detect from which code artifact the bugs were stemming from, and how the problematic code artifact should be restructured to achieve the desired requirement. The sources from which the bugs could have been stemming from were limited, so it didn't take longer than a day to track them down and fix them. I verified the functionality through extensive testing, though it's difficult to conclude whether all bugs were found, given the complexity of interactions that are possible within a game. At the very least, no bugs manifested in the typical gameplay scenarios I tested. I don't mean to imply that the TLLM wasn't useful at all in implementing and integrating the Combo Spell system, but merely to highlight the fact that relying solely on the TLLM would have led to a poor result.

The code generated by the TLLM was sometimes inefficient or otherwise contained needless redundancies, as in the following snippet:

```
if equipped_spell is ComboSpell:
    equipped_spell.cast(self) # signals combo injection
elif equipped_spell is IncantationSpell:
    equipped_spell.cast(self) # starts incantation
else:
    equipped_spell.cast(self) # instant cast
```

A code snippet generated for SpellCaster.gd

It's clear that the TLLM was more focused on implementing raw functionality over anything else. However, it also highlights the fact that the TLLM possesses no real reasoning or understanding of context. If one were to opt for a purely vibe coding-based method, this would lead to confusing, redundant, and buggy code. The TLLM tends to be useful for debugging only if the bug is obvious, or if the developer is able to pinpoint the bug's origin but is unsure of how to resolve it.

It's also worth reflecting on what is actually happening when, for example, the TLLM recommends a design for a system like it did for the SpellCaster. It's understood that an LLM is merely highly complex data retrieval system that returns information based on the input tokens [18]. Therefore, output from an LLM should not be thought of as "the LLM generated this design" but rather as "the LLM retrieved this design from its dataset because it is how other developers typically design these systems, so perhaps I should too." Anthropomorphic language is useful to describe LLM behaviour concisely, but it's important that we don't lose sight of what's actually happening under the hood.

3.1.1 Takeaways from Case Study 1

From the implementation of the PA, some general lessons can be extracted. The TLLM can be thought of as a highly-curated document retrieval system, allowing for very specific insights on your CBSs. However, it is incapable of reason or assessing context, and thus output must be critically assessed and sometimes altered by the developer.

Another situation in which the TLLM excels is when there exists an easy solution to a problem of which the developer simply isn't aware. Rather than spend time searching for such a solution, a process which sometimes has no clear result, one can prompt the TLLM and be directly given a solution. If an easy solution does not exist, then the developer is merely back to square one and can rely on other methods of problem-solving. The most banal but most frequently encountered manifestation of this situation was syntax errors. The TLLM tended to fail here only if the code it generated was for an older code version.

Tangentially related is the TLLM's ability to lay foundational groundwork for systems and system artifacts, also called scaffolding. Rather than spend time conceptualizing how largely boiler-plate systems should manifest and connect to one another, one can lay the burden of such work on the TLLM and instead focus more on fleshing out project-specific artifacts or nailing down functionality based on testing feedback. Furthermore, assuming one has kept software engineering principles in mind during development and has created or facilitated the creation of well-designed and modular code, the TLLM does a good job of extending, augmenting, or otherwise refactoring designs as needed.

Finally, the WPPM can serve as a tool for learning, assuming one has enough of a coding and software engineering background and is aware of situations in which the TLLM is hallucinating. However, without such a background, or with continued over-reliance on LLMs, it's likely that the average developer will lose their ability to recognize such hallucinations. With its diverse dataset, the TLLM can retrieve and generate good insights for best practices, giving multiple approaches to solving a problem and the strengths and weaknesses of each one. Again, the quality of the output is dependent on the developer's ability to understand and verify the contents of the output; however, as seen with the refactoring of the attack system, proposed solutions generated by the TLLM can be reinterpreted and re-implemented by the developer to solve other problems. From my experience, it didn't take long to understand and verify the correctness of the generated code; I already had an understanding of what the code should be doing, and could easily assess the correctness by testing the game.

3.2 Case Study 2: System-Controlled Artifacts

Initially, there were some difficulties when it came to the foundational implementation of the Boss Enemy. The difficulties were partly due to my lack of experience with Godot, and thus my being unsure of how a Boss Enemy should be structured architecturally. This led to some implementation details, such as a superfluous script, that ended up being scrapped later. The TLLM also got a bit overexcited, to use anthropomorphic language, with regards to the implementation of the Boss Enemy, adding additional details that were not present in the original design. For example, the TLLM added an elemental theme to the Boss Enemy phases, which had little to no basis in the input prompt aside from the Boss Enemy attacks' being themed around fire, ice, and lightning.

However, once the architecture was more cemented, and I had a better understanding of how the different relevant system components interacted with each other, the development process became much smoother. Additionally, the TLLM provided a good deal of ancillary information regarding the design, such as example file structure setups, as well as providing further information on what each part of the generated design is responsible for. This can serve as a sanity check to ensure each component is consistent with both the outlined design and the expectations for what the CBS should look like. The TLLM still hallucinated, but I had enough domain knowledge to pick it out and account for it in the implementation. It's worth noting that in the event of hallucinated output, pointing out the inconsistencies in the hallucination in a follow-up prompt can lead to a correctly revised output; at the very least, it can serve as another layer of sanity checking for the developer.

There was little difficulty in the implementation of a certain game effect, be it a Boss Enemy attack or a PA spell, through the WPPM. Both Boss Enemy attacks and PA spells largely consist of a sprite animation and a hitbox, with a script controlling the playing of certain animations or the activation of a hitbox. While some effects have some additional nuance, like the Celestial Primate's Ice Shock attack slowing the PA's movement for a short period, they didn't often cause much difficulty in implementation.

When there was some difficulty, it was for effects with a high amount of complexity, like the Chaos Zone Mid-Combat Event. With such complexity, the TLLM did not generate a correct design in one-shot; however, I was able to work through such difficulties and implement the Chaos Zone Mid-Combat Event, both using the TLLM and my own development experience to achieve the desired result. In fact, there are no proposed Boss Enemy attacks or PA spell effects that I found myself incapable of implementing through use of the WPPM.

3.2.1 Takeaways from Case Study 2

The previous case study as well as this one established that the WPPM is quite good for scaffolding any given code artifact. Furthermore, I had the impression during development that, even in the case in which the TLLM generated only a partially correct system, it was easier to debug such a partially correct system than it would have been to write the same system myself and debug it. This impression comes from comparing my development of this game app with the game development I've done in the past; however, it's unclear whether such an impression is rooted in any measured gain in productivity. It's also possible that any supposed increase in productivity would be unique to the domain of game development, which tends to reuse and reinterpret code modules, and thus would be well positioned for the strengths an LLM would bring.

This remains separate from the time it takes to find the source of a bug in the first place; it's possible that finding the source of a bug in code generated by the TLLM would take more time to track down, as opposed to tracking down the source of a bug in code written manually by the developer. Furthermore, developers, as human beings, are known to write similarly-formed programs and thus introduce similar kinds of bugs [23, 27, 9]. It is unclear what types of bugs would typically manifest in the code generated by LLMs, and that uncertainty could cause further difficulties in tracking down the source of such bugs. The bottleneck appears to be in the verification of code generated by an LLM, and in the difficulty in performing such verification in the relevant software development domain [11]. There's an argument to be made that under the WPPM, the developer is more actively

primed to notice, fix, and test for bugs than the developer typically would be in a standard development process. The counter-argument is that, given the generated code is written by no one, it would take longer to understand the code such that it can be debugged than it would have been to write it oneself and debug it from such a perspective. Without performing a cost analysis between the two methods, it's difficult to come to a conclusion either way.

As mentioned above, there weren't any ideas that were conceived that I wasn't able to bring to fruition. With enough knowledge of programming principles, the WPPM can serve to lower the knowledge barrier that more complex interactions would demand. Furthermore, the ancillary information provided by the TLLM can serve as a loose sanity check; firstly that the TLLM is doing what the developer expects, and secondly that the design being outlined is in line with software engineering principles. If either check fails, prompting the TLLM about such inconsistencies can serve to generate a corrected design; at the very least, the developer can get a further grasp on how different code artifacts are typically designed or how such code artifacts should interact with each other.

It isn't always possible to communicate a full vision immediately, especially when lacking a certain degree of knowledge on how code artifacts will be set up or how they will interact. Coupled with the fact that LLMs are incapable of reasoning, it is very important that a developer does not treat the TLLM as any kind of collaborator. It's tempting to do so, given the human tendency to anthropomorphize, but it can be a slippery slope leading to holes out of which one may not find a way; at worst, you compromise on the vision outlined by the SRS and are unable to reconcile the requirements with what has been implemented.

3.3 Case Study 3: Top-Level Artifacts

As a primary UI artifact, the HUD is a front-facing visual artifact more than it is a code artifact. This is an area the TLLM struggled the most, as LLMs are primarily language models that struggle with graphical understanding. Furthermore, the disconnect between the platform hosting the TLLM and the game engine itself exacerbates the problems, due to the TLLM's lack of context. In previous cases, unmentioned until now, the TLLM would use random location coordinates for the spawning of code artifacts, given it had no reference for where such code artifacts should be placed. This typically wasn't too much of a problem, as the location of a code artifact was largely inconsequential compared to other implementation details.

The problem is further exacerbated with a primarily visual artifact like the HUD, with the TLLM unable to provide guidance that functioned well out-of-the-box, and required much more manual work by myself. Unfortunately, the overall process was not as smooth as it was with other facets of development.

There was still some benefit to be gained from the TLLM with regards to general setup and indicating to me what Nodes would be useful in the implementation. The TLLM remained useful in the implementation of the code-based artifacts of the HUD, or otherwise augmenting such code-based artifacts, such as implementing logic to highlight the currently selected spell on the HUD interface after the spell-casting system was refactored.

3.3.1 Takeaways from Case Study 3

It's clear that a primary issue for the TLLM is the implementation of visual artifacts within the system. This is likely partly a symptom of the way the WPPM interfaces with two disconnected systems: the ChatGPT webpage, and the Godot game engine. However, it does provide further evidence for how exactly the TLLM lacks reason, and more concretely exposes the insidious nature behind the technology: the priority to generate a response that looks good, but isn't necessarily correct. This wouldn't be the first case of the TLLM prioritizing a correct-looking response over a logically correct response during development; however, it is enough of a banal case to bring the issue into focus. Despite this, the TLLM remained useful scaffolding code artifacts, consistent with other facets of the project.

This also further reinforces the idea that LLMs serve as good vehicles for code refactoring, so long as the developer understands the impetus behind the refactor and is aware of how the refactoring will take form. Ultimately, a refactor is a process that seeks to preserve as much of the original functionality as possible, ideally all of it. This implies that the desired functionality is already implemented, and thus the use case is well-positioned for an LLM as it provides clear bounds within which to work. This can similarly be mobilized as an argument for why the TLLM is adept at augmenting or otherwise extending the functionality of a code artifact. The issue that comes into focus is whether or not developers are aware of the reasons and method behind performing a refactor; the assumption that they do may hold less and less weight, as over-reliance on LLMs atrophies the problem-solving skills of such developers.

Chapter 4

Discussion

4.1 Reflections on the Thesis Research

This thesis has provided evidence of the feasibility for a TLLM to facilitate game development using an SRS as input through the WPPM. The wealth of information from both the SRS and from the progressive nature of the WPPM leads to high-quality and functionally accurate code. The results reinforce how prompt engineering in this domain is misguided from principle. The SRS, if done properly, should already cover the full scope of what the code should accomplish, and thus should be all the data an LLM would need to generate this code. Therefore, it's unclear to me what else would be necessary to persuade an LLM to generate the code.

From the case studies, it's clear that the TLLM works best to solve problems related to searching through a library. Much of the benefit of using the WPPM and its TLLM come from the TLLM's retrieval of reusable modules that can easily be tweaked, either by the TLLM itself or the developer, to suit the context of the requirement. If such an artifact cannot be retrieved, then the developer is back to square one with little lost, and can manually work through solving the requirement. This can be as simple as retrieving information related to syntax, to more complex reuses of modules for new objects that share much of the same structure.

The TLLM also worked well at scaffolding, quickly generating boilerplate code and allowing for platforming to be done with little effort. This also largely boils down to searching through a library and retrieving relevant code artifacts, and thus is well suited to the strengths of the TLLM. The general theme that forms from these takeaways is that

delegating more menial, rote tasks onto the TLLM leaves the developer free to focus on the more creative facets of the project. In addition, the developer is also less beholden to their skill barrier when it comes to conceptualizing and implementing unique creative ideas; so long as the developer has a good background in software development principles and is able to work through how the idea would manifest, the TLLM can serve to guide them the rest of the way. This argument presumes that the idea the developer has is itself feasible, and that the TLLM's data contains relevant code modules that can be used to implement the idea.

Whenever the TLLM did not provide suitable code to fulfill a requirement, I saw that it was better for me to write the code for the requirement manually. This could be done either from scratch, or from the already generated code artifacts, depending on the quality of the generated output and on the broader context of the implementation. Thus, requirements are not all you need to persuade a TLLM to generate an executable that fulfills those requirements. There are many creative steps and engineering decisions that are necessary to the production of useful output from a TLLM. In other words, if you were to give a TLLM the whole SRS and told it to generate code for a complete Godot game app project as specified in the SRS, you'd do so with the expectation that it would fail. As an exercise, I did just that, and gave my SRS and UCs to a new manifestation of the TLLM running on GPT-5.3, and asked the TLLM to generate code that fulfills the SRS and UCs. The TLLM just regurgitated the SRS and UCs back to me. So, I asked the TLLM explicitly to generate code that fulfills all UCs. It failed. Figure 4.1 shows the TLLM's response to my last request.

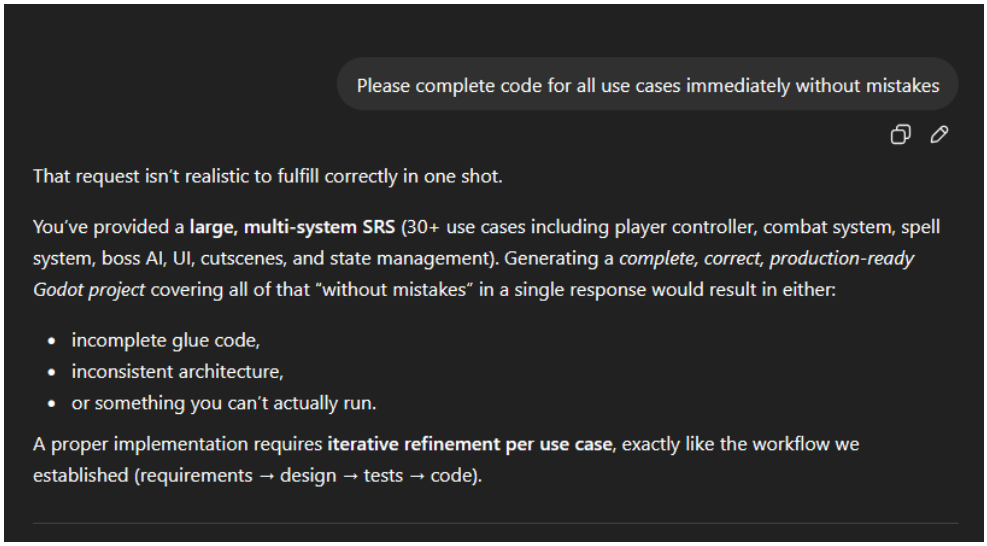


Figure 4.1: The TLLM's Response to the Request

Domain expertise and expertise in requirements and software engineering remain necessary to the development process, even with LLM assistance.

4.2 Reflections on the Broad Impact of LLMs

This ties into a more general concept that has been discussed with regards to the new software development paradigm brought in by the advent of LLMs: the democratization of code [5, 16]. As posited, proponents of the concept claim that software development is now at the grasp of anyone, assuming that one is able to render their software idea in natural language. While this may sound handsome in concept, when considered critically it falls apart, given programming and software development as a whole are extremely complex as mediums. There were many examples in the three case studies of output that were almost correct, but required additional developer oversight to get all the way there. Even more critical was the fact that the diagnoses of many esoteric bugs, such as those tied to race conditions, depended on my domain expertise as a software developer. You won't find many people calling for the democratization of medical practice, as it is understood that medicine is a highly technical field that requires years of study; is software development not such a technical field?

It's worth reflecting on how the output from LLMs is structured and the tone they take.

LLM output tends to take an overly cheery tone, praising your ideas and even sometimes adding more onto them without question–in other words, if you treat the TLLM like a collaborator, you'll simply have a sycophant without capacity for reason plugged directly into your codebase [17]. You must be capable of critically evaluating the quality of the output across all facets of the relevant domain. Unfortunately, there is evidence that many developers are not verifying the correctness of generated code, despite understanding that generated code often contains bugs [11]. The generated suggestions from the TLLM, or any LLM, are only as good as how you, the developer, are able to interpret them. The burden of conceptualization is placed uniquely on the human; such a relationship can scarcely be called co-creation.

The spectre of vibe coding has haunted the bulk of this thesis, so let's address it. Despite research claiming that this method is supposedly the way of the future, I maintain that vibe coding is completely infeasible as a method for software development. The fundamental issue goes back to the basics of how language and communication itself manifest in the material world: natural language exists as a proxy for meaning, and its manifestation as a proxy is what leads to ambiguity. On the other hand, high-level coding languages exist as a proxy for lower-level machine processes.

This fundamental issue natural language has in delivering meaning explodes in complexity and scale when parsing it through an LLM, which then reduces it even further to a mathematical proxy, to then be reformed in this case into a high-level coding language. This is in part what leads to hallucinations, feeding into technical debt and security risks; adding further frameworks to try and correct for the symptoms arising from this fundamentally flawed epistemic pipeline is a fool's errand, and could instead be solved simply by a developer doing the job that developer was hired to do.

4.3 The Dangers of LLM-Based Coding

This focus on realizing a vibe or directing an LLM using natural language and examples is just obfuscating what software development actually is: the translation of requirements into a CBS that achieves those requirements. Furthermore, it's clear LLMs cannot understand what a vibe is, and besides that, what should be fulfilled is not some nebulous vibe but instead the requirements. The success derived from similar explorations of the capacity of LLMs as code generation drivers is due to the fact that, fundamentally speaking, LLMs are good at searching for and retrieving relevant code modules that can often fulfill many of the requirements of the CBS [1]. Anthropomorphism clouds this fact, and leads to more

flowery conceptualizations as to what the relationship between the developer and the LLM is.

Despite the wealth of evidence, both academic and anecdotal, arguing against brazen practices such as vibe coding, the academic space continues to approach LLMs with wide-eyed optimism [5, 15, 12, 16]. Rather than challenge the assertions pushed by the corporate sector, who blatantly have incentive to present their technology as messianic, many researchers have staunchly fallen in line with the narratives pushed by such bodies, both in language and in how they conceptualize the future of the technology. Vibe coding itself was presented in a tweet by one such corporate leader, stating that it is "a new kind of coding where you fully give in to the vibes, embrace exponentials, and forget that the code even exists "[10]. Even without verifying the claim, anyone with any programming experience should read this proposal as infeasible and ludicrous in nature. Should we as researchers not be more careful when it comes to embracing new technologies, and look to understand them as they manifest in practice under a worst-case scenario, rather than build frameworks and theories assuming the best-case?

Chapter 5

Conclusions

5.1 Limitations

This thesis is an experience report describing an LLM-assisted development of a substantial game app. The game app is small enough to be developed by one person in the time normally allocated to a Master's thesis, but large enough to require all of that time. It represents a fully complete game app demo. In other words, were development to continue on this game app, the components that are already completed would not be altered.

This experience report describes only one development process. The conclusions are drawn from that one development and cannot be generalized. However, this report does contribute to a growing body of experience reports about the developments of variously-sized systems. Any generalization can only be drawn when commonalities are observed in enough experience reports. Furthermore, the method used in this thesis has quickly become outdated, as the methodology concerning the best uses and configurations for LLMs is progressing faster than research can be done studying the capacity of LLMs themselves. While this thesis uses a method that had become outdated during the experience, the insights concerning LLMs themselves are still prescient.

The TLLM was built using the GPTs function provided by ChatGPT and was created by Bingyang Wei; however, my personal manifestation of the TLLM was running on the free plan of GPT-4 hosted on the ChatGPT website. It's unclear if the manifestation was forcibly upgraded or not during development; the platform does not provide any real way to check. Prompting the TLLM itself indicated that it was using GPT 5.1 at time of writing, but this was contradicted by a tooltip on the ChatGPT website stating that the

chat will continue to use the TLLM's old version, GPT 4.0, and a new chat must be started to use the new version. It's also unclear what ramifications the different version upgrades have on the quality of output, or if the upgrade corrupted the context of TLLM.

On the latter point, after a certain point late in development, the TLLM lost all earlier context, including the originally input SRS as well as the UCs themselves. Given the volatile nature of the technology, subject entirely to the whims of OpenAI, it's very difficult to determine if this loss of context was a consequence of the limitations of the technology itself, or if it was due to the forced upgrade of the TLLM, or any number of other possibilities. The way in which these technologies are platformed means that there's not only a black-box surrounding the technology itself, but also around the bureaucratic and corporate structures that support it.

Aside from these foundational issues, there were also some limitations in terms of workflow. I mentioned before that the disconnect between the ChatGPT platform and the Godot game engine caused some minor problems, so I won't reiterate that point here. Otherwise, interfacing with the TLLM using the chat-style interface is quite cumbersome; for example, despite making minor changes to the generated code being possible, doing so would require another conversation turn and regeneration of code as well as any other additional text artifacts. This problem is exacerbated by the fact that, as chat messages accumulate, the webpage begins to lag quite heavily. This could be a consequence of using the free plan of the ChatGPT platform, but this was not verified.

Finally, it's worth noting that the original WPPM includes a final step for generating test cases to confirm functionality. I did not make use of this function because my focus was on exploring and assessing the feasibility of the WPPM. I opted to perform my own manual testing, which I believe was still effective in assessing the validity of both the code I wrote and the code generated by the TLLM.

5.2 Future Work

Although this thesis served to present a proof-of-concept of leveraging LLMs for game development, there remain many opportunities for future evaluation and further development of tools based on these results. Taking Wei's work and this thesis in tandem can provide us with a clearer spectrum by which to understand the capacity of LLMs and the WPPM. However, more specialized systems, such as real-time systems, likely fall out of this spectrum and would require their own testing. Other genres in the gaming domain may also have their own unique complications, and thus further iteration in this domain would clarify the limits of the TLLM and the WPPM's capacity.

Issues that manifested during development were commonly tied to the use of the OpenAI platform, both the cumbersome nature of interfacing with the TLLM through it, as well as other more systemic issues such as the loss of context and possibly forced version upgrades. I envisage a system built specifically for the WPPM, with its own TLLM: this system would address the primary pain points that were encountered in development, such as allowing for the making of minor changes without necessitating a full conversation turn. This system could also facilitate the creation of the SRS itself, streamlining development and allowing for a more cemented context for the TLLM to draw from.

Another possible direction would be to build a system that integrates into a game engine like Godot, allowing for more direct and accurate recommendations from the TLLM. The system could highlight specific artifacts visually rather than referring to them rhetorically, or even act directly to implement certain artifacts with the consent of the developer. The system would also have more context with regards to the visual artifacts of the game, and could provide more accurate output accordingly.

The most important future effort in my view is to aim to combat the prevailing narratives surrounding LLMs, and seek to foster a body of research focused on evaluating such systems critically. These technologies are only as useful as they manifest practically, and thus it's essential that we approach research from an angle of practicality. Doing so will give us more insight on how we can integrate LLMs into our development process to both improve the efficiency of development, as well as the quality of output. With that in mind, performing a cost analysis of manual development against the WPPM will help us better understand whether TLLM-based methods provide any material benefit in the domain of game development, and is an important next step.

5.3 Conclusion

This thesis provides clear evidence for the feasibility of the WPPM for game development, using a well-articulated SRS document as input for a TLLM designed for software development. The SRS encapsulates the full scope of the CBS's functionality, and thus is well-positioned as input for LLMs, which have shown potential for translating such requirements into working code. Case studies encapsulating the development of different types of code artifacts support these assertions, demonstrating the practicality and effectiveness of the WPPM. Takeaways from the results of the thesis work serve to help us better evaluate the strengths and weaknesses of LLMs, and combat the dominant narratives surrounding these systems. Most importantly, this thesis reinforces the importance of fleshed out and well-articulated SRS in software development.

Future work will seek to perform a cost analysis of manual development against the WPPM, and assess the strengths of both methods across different code artifact types. While the silver-bullet fascination with LLMs hasn't yet passed, it's important that we continue to assess this technology critically so we can be left with a truly useful aluminum bullet.

References

[1] Daniel M. Berry. The True Cost of AI Assistance to Programming of Software. *IEEE Software*, 43(2):130–135, 3 2026.

[2] Som Biswas. Role of ChatGPT in computer programming. *Mesopotamian Journal of Computer Science*, 2023:9–15, 2 2023.

[3] Brooks. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, 4 1987.

[4] Margarida Fortes-Ferreira, Md Shadab Alam, and Pavlo Bazilinskyy. Vibe Coding in Practice: Building a Driving Simulator Without Expert Programming Skills. *AutomotiveUI Adjunct '25*, pages 60–66, 9 2025.

[5] Akhilesh Gadde. Democratizing Software Engineering through Generative AI and Vibe Coding: The Evolution of No-Code Development. *Journal of Computer Science and Technology Studies*, 7(4):556–572, 5 2025.

[6] April M. Grow and Foaad Khosmood. Chatgpt gamejam: Unleashing the power of large language models for game jams. In *Proceedings of the 7th International Conference on Game Jams, Hackathons and Game Creation Events*, ICGJ '23, page 51–54, New York, NY, USA, 2023. Association for Computing Machinery.

[7] Mehmet Safa Gökalp and Ayça Kolukısa. Chatgpt based best practices for pair programming. In *2025 10th International Conference on Computer Science and Engineering (UBMK)*, pages 254–259, 2025.

[8] Md. Asraful Haque. Llms: A game-changer for software engineers? *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 5(1):100204, 2025.

[9] Abram Hindle, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. On the naturalness of software. *Commun. ACM*, 59(5):122–131, April 2016.

[10] Andrej Karpathy. Post on "vibe coding", 2025.

[11] Nicole Kobie. So much for 'trust but verify': Nearly half of software developers don't check AI-generated code.... 1 2026.

[12] Madhava Krishna, Bhagesh Gaur, Arsh Verma, and Pankaj Jalote. Using LLMs in Software Requirements Specifications: An Empirical Evaluation. *unpublished*, pages 475–483, 6 2024.

[13] Mohammadamin Madani, Ksenia Neumann, Abdulrahman Nahhas, Maria Chernigovskaya, Damanpreet Singh Walia, and Klaus Turowski. Towards the integration of large language models into the software development life cycle: A systematic literature review. In *2025 3rd International Conference on Foundation and Large Language Models (FLLM)*, pages 773–781, 2025.

[14] Stephane Maes. The gotchas of AI coding and vibe coding. It's all about support and maintenance. *Zenodo (CERN European Organization for Nuclear Research)*, 4 2025.

[15] Jacqueline Mitchell and Yasser Shaaban. Position: Vibe Coding Needs Vibe Reasoning: Improving Vibe Coding with Formal Verification. *LMPL '25*, pages 84–90, 10 2025.

[16] Veronica Pimenova, Sarah Fakhoury, Christian Bird, Margaret-Anne Storey, and Madeline Endres. Good Vibrations? A qualitative study of Co-Creation, communication, flow, and trust in vibe coding. *arXiv (Cornell University)*, 9 2025.

[17] Rafael Ris-Ala, Gonçalo Gonçalves, Leonardo S. Lopes, Tiago F. Dantas, Dennis Paulino, André T. Netto, Diogo Guimarães, Artur Rocha, Adriana S. Vivacqua, and Hugo Paredes. Do llms tell us what we want to hear? investigating confirmation bias in ai responses to health queries. In *2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4338–4341, 2025.

[18] Cole Stryker. Large Language Models, 12 2025.

[19] Dan-Alexandru Ternar, Alena Denisova, João Miguel Cunha, Annakaisa Kultima, and Christian Guckelsberger. Generative ai in game development: A qualitative research synthesis. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA, 2026. Association for Computing Machinery.

[20] Jonathan Ullrich, Matthias Koch, and Andreas Vogelsang. From requirements to code: Understanding developer practices in llm-assisted software engineering. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, pages 257–266, 2025.

[21] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, 4 2022.

[22] Pau Vivas Zamora. *AI-assisted game development: designing and prototyping core gameplay systems.* Treball final de grau, UPC, Centre de la Imatge i la Tecnologia Multimèdia, 2025.

[23] Chuanqi Wang, Yanhui Li, Lin Chen, Wenchin Huang, Yuming Zhou, and Baowen Xu. Examining the effects of developer familiarity on bug fixing. *Journal of Systems and Software*, 169:110667, 5 2020.

[24] Shangwen Wang, Mingyang Geng, Bo Lin, Zhensu Sun, Ming Wen, Yepang Liu, Li Li, Tegawendé F. Bissyandé, and Xiaoguang Mao. Natural Language to Code: How Far Are We? *ESEC/FSE '23*, pages 375–387, 11 2023.

[25] Bingyang Wei. Requirements are All You Need: From Requirements to Code with LLMs. *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pages 416–422, 6 2024.

[26] Ziyao Zhang, Chong Wang, Yanlin Wang, Ensheng Shi, Yuchi Ma, Wanjun Zhong, Jiachi Chen, Mingzhi Mao, and Zibin Zheng. LLM Hallucinations in Practical Code Generation: Phenomena, mechanism, and mitigation. 2(ISSTA):481–503, 6 2025.

[27] Wenhan Zhu and Michael W. Godfrey. Mea culpa: How developers fix their own simple bugs differently from other developers. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 515–519, 2021.

[28] Yilun Zhu, Joel Ruben Antony Moniz, Shruti Bhargava, Jiarui Lu, Dhivya Piraviperumal, Site Li, Yuan Zhang, Hong Yu, and Bo-Hsiang Tseng. Can large language models understand context? *arXiv (Cornell University)*, 2 2024.

APPENDICES

Contributed Artifacts

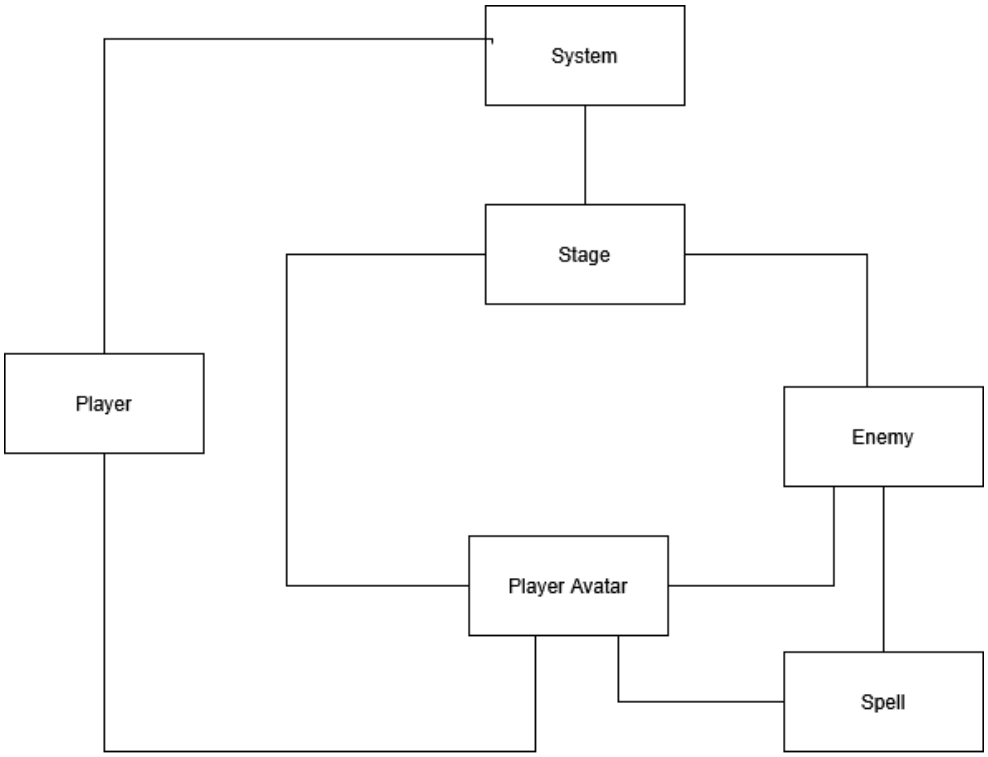
- Game App Source Code → https://github.com/aelshatshat/Ahmed-E-Master-s-Thesis-My-Adventure
- Game App Assets → https://github.com/aelshatshat/Ahmed-E-Master-s-Thesis-My-Adventure/tree/main/my-adventure/sprites
- TLLM Chat History → https://github.com/aelshatshat/Ahmed-E-Master-s-Thesis-My-Adventure/blob/main/docs/Facilitating%20Game%20Development%20From%20Requirements%20to%20Code%20using%20LLMs%20Chat%20History.html
- Full SRS → https://github.com/aelshatshat/Ahmed-E-Master-s-Thesis-My-Adventure/tree/main/docs/SRS

Appendix A contains the primary SRS documents: the Domain Model, the Use Case Model, and the Use Case Document.

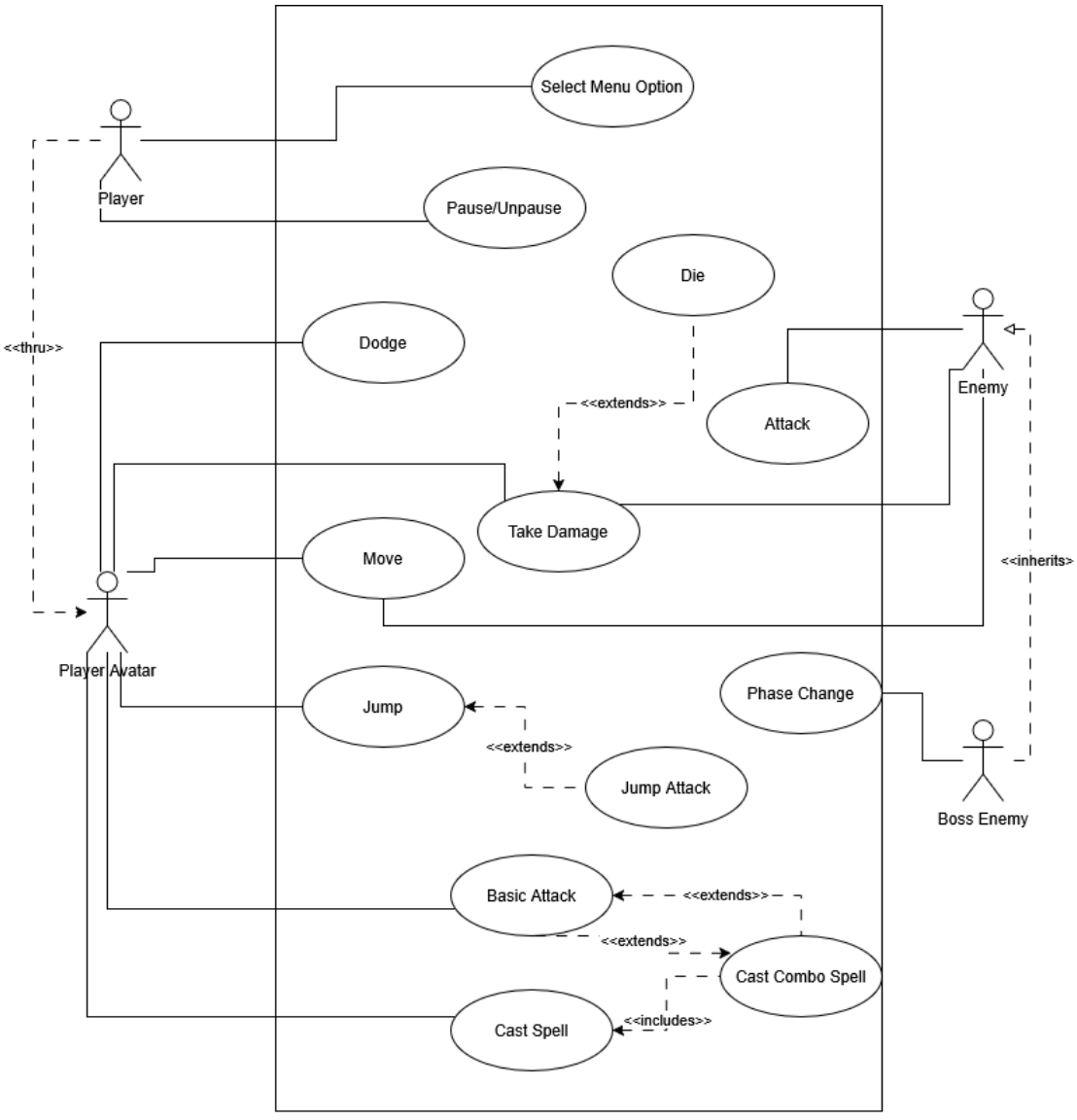
Appendix A

SRS Documents

A.1 The Domain Model



A.2 The Use Case Model



A.3 The Use Case Document

UC ID and Name:	UC-1: Player Avatar Move Left and Right
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the A key or the D key
Description:	The player wants the Player Avatar to move in a horizontal direction
Preconditions:	PRE-1. Player Avatar is in a moveable state
Postconditions:	POST-1. The Player Avatar moves in the appropriate direction (left for A, right for D) POST-2. A running animation plays for the Player Avatar
Main Success Scenario:	1. The player presses the A key or the D key 2. The Player Avatar moves left or right, depending on the direction 3. Use Case ends
Extensions:	**1a. Multiple Input rule violation:** 1a1. The Player Avatar does not move (both A and D are pressed). 1a2. The player releases one of the two keys, and the Player Avatar moves in the direction of the still pressed key **2a. Blocked direction:** 2a1. The Player Avatar moves in place, blocked by the environment

UC ID and Name:	UC-2: Player Avatar Jump
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar

Trigger:	The player pushes the SPACE key
Description:	The player wants the Player Avatar to move in a vertical direction
Preconditions:	PRE-1. Player Avatar is in a moveable state
Postconditions:	POST-1. The Player Avatar moves upwards POST-2. The Player Avatar changes to a jump animation
Main Success Scenario:	1. The Player presses the SPACE key 2. The Player Avatar jumps upward 3. The Player Avatar changes to a jumping animation 4. Use Case ends
Extensions:	**4a. Player Avatar Land** 4a1. The Player Avatar falls and plays a falling animation 4a2. The Player Avatar lands on the ground and ceases to fall 4a3. The Player Avatar changes to an idle animation 4a4. Use Case ends **4b. Player Avatar Double Jump** 4b1. The Player presses the SPACE key while still airborne after jumping once 4b2. The Player Avatar jumps upward once more 4b3. The Player Avatar plays its jumping animation again 4b4. Use Case continues to 4a.

UC ID and Name:	UC-3: Player Avatar Sword Attack
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Left-Mouse Button
Description:	The player wants the Player Avatar to attack using the sword
Preconditions:	PRE-1. Player Avatar is in a moveable state
Postconditions:	POST-1. The Player Avatar attacks using a sword

	POST-2. The Player Avatar changes to sword attack animation
Main Success Scenario:	1. The player presses the Left-Mouse Button 2. Player Avatar animation changes to the first sword attack animation 3. The Player Avatar attacks with a sword 4. Use Case ends
Extensions:	**4a. Combo Attack 1** 4a1. The player presses the Left-Mouse Button again within 0.2 seconds of the animation finishing 4a2. The Player Avatar changes to the second sword attack animation 4a3. The Player Avatar attacks with the second sword attack in the combo **4b. Combo Attack 2** 4b1. The player presses the Left-Mouse Button again within 0.2 seconds of the animation finishing 4b2. The Player Avatar changes to the second sword attack animation 4b3. The Player Avatar attacks with the second sword attack in the combo 4b4. The player presses the Left-Mouse Button for the third time within 0.2 seconds of the animation finishing 4b5. The Player Avatar changes to the third sword attack animation 4b6. The Player Avatar attacks with the third sword attack in the combo **4c. Combo Attack 3** 4c1. The player presses the Left-Mouse Button again within 0.2 seconds of the animation finishing 4c2. The Player Avatar changes to the second sword attack animation 4c3. The Player Avatar attacks with the second sword attack in the combo 4c4. The player presses the Left-Mouse Button for the third time within 0.2 seconds of the animation finishing

	4c5. The Player Avatar changes to the third sword attack animation 4c6. The Player Avatar attacks with the third sword attack in the combo 4c7. The player presses the Left-Mouse Button for the fourth time within 0.2 seconds of the animation finishing 4c8. The Player Avatar changes to the fourth sword attack animation 4c9. The Player Avatar attacks with the fourth sword attack in the combo

UC ID and Name:	UC-4: Player Avatar Spinning Jump Attack
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Left-Mouse Button while in a jump state
Description:	The player wants the Player Avatar to attack using the sword in the air
Preconditions:	PRE-1. Player Avatar is in a jump state PRE-2. Player Avatar is in a moveable state
Postconditions:	POST-1. The Player Avatar attacks in the air using a sword spin attack POST-2. The Player Avatar changes to a cycling sword spin attack animation
Main Success Scenario:	1. The Player presses the SPACE key 2. The Player Avatar jumps into the air 3. The Player Avatar changes to a jumping animation 4. The Player presses the Left-Mouse Button 5. The Player Avatar does a spinning jump attack 6. The Player Avatar changes to a spin attack animation 7. The Player Avatar touches the ground 8. Use Case ends

Extensions:	

UC ID and Name:	UC-5: Player Avatar Cast Spell
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Right-Mouse Button
Description:	The player wants the Player Avatar to attack using a spell
Preconditions:	PRE-1. Player Avatar is in a moveable state PRE-2. Player Avatar's Mana is above the spell's Mana cost PRE-3. Active spell's cooldown timer is at 0
Postconditions:	POST-1. The Player Avatar casts a spell POST-2. The Player Avatar changes to spell cast animation POST-3. Active spell is on cooldown for its cooldown duration
Main Success Scenario:	1. The Player presses the Right-Mouse Button 2. The Player Avatar casts a spell 3. The Player Avatar plays the appropriate spell-casting animation based on the spell type 4. Player Avatar Mana resource is reduced by the amount required by the spell 5. Mana Resource bar is updated 6. Use Case ends
Extensions:	**3a. Combo Spell Injection** PRE-3 Player Avatar is currently in UC-3 PRE-4 Currently active spell is a Combo spell 3a1. The Player Avatar casts a Combo spell at the next combo step, based on the last combo step reached in UC-3 3a2. Player Avatar plays appropriate animation for Combo spell step 3a3. Use Case continues at 4. in the Main Success Scenario **3b. Incantation Spell Cast**

	PRE-3. Currently active spell is an Incantation Spell 3b1. The Player Avatar movement is locked 3b2. The Player Avatar changes to a casting animation 3b3. A cast time progress bar appears above the Player Avatar 3b4. Cast time progress bar is filled 3b5. Player Avatar casts spell 3b6. Use Case continues at 4. in the Main Success Scenario

UC ID and Name:	UC-6: Player Avatar Dodge Cast
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Shift Button
Description:	The player wants the Player Avatar to dodge and avoid damage for a time
Preconditions:	PRE-1. Player Avatar is in a moveable state PRE-2. Dodge Cooldown is currently at 0
Postconditions:	POST-1. The Player Avatar moves quickly in the facing direction POST-2. The Player Avatar changes to a dodge animation POST-3. The Player Avatar is invulnerable to damage for 0.5 seconds POST-4. Dodge Cooldown is reset
Main Success Scenario:	1. The Player presses the Shift-Button 2. The Player Avatar dodges in a direction 3. The Player Avatar changes to a dodge animation 4. The Player Avatar is invulnerable for the duration of the dodge animation 5. Use Case ends
Extensions:	

UC ID and Name:	UC-7: Player Avatar Take Damage
Created By:	
Date Created:	
Primary Actor:	Player Avatar
Secondary Actors:	Enemy
Trigger:	The player gets struck with an enemy attack
Description:	The Player Avatar takes damage from an enemy attack
Preconditions:	PRE-1. Player Avatar is in a vulnerable state PRE-2. Enemy health is above 0 PRE-3. Player health is above 0
Postconditions:	POST-1. The Player Avatar takes damage POST-2. The Player Avatar plays a take damage animation POST-3. The Player Avatar is invulnerable to damage for 1 second POST-4. Health resource bar is updated with new value
Main Success Scenario:	1. The Player Avatar gets hit by an enemy attack 2. The Player Avatar Health decreases based on the damage value of the enemy attack 3. The Player Avatar is in an immovable state for 0.2 seconds and is an invulnerable state for 0.5 seconds 4. The Player Avatar changes to a taking damage animation 5. Player regains control of Player Avatar 6. Player Avatar loses invulnerability 7. Use Case ends
Extensions:	**3a. Player Avatar Dies** 3a1. GO TO UC-8 3. **4a. Player Avatar Gets Debuffed** 4a1. The Player Avatar gains an additional negative effect from the enemy attack **4b. Knockback** PRE-4. Enemy attack has a knockback value 4b1. Player Avatar is pushed in the knockback direction based on the knockback value of the attack

UC ID and Name:	UC-8: Player Avatar Dies
Created By:	
Date Created:	
Primary Actor:	Player Avatar
Secondary Actors:	Enemy
Trigger:	The Player Avatar gets struck with an enemy attack and loses all remaining Health points
Description:	The Player Avatar dies after taking too much damage
Preconditions:	PRE-1. Player Avatar is in a vulnerable state PRE-2. Enemy health is above 0 PRE-3. Player Avatar Health is above 0
Postconditions:	POST-1. Player Avatar Health is less than or equal to 0 POST-2. The Player Avatar changes to a death animation POST-3. The Player Avatar is in an immovable state POST-4. The player is in a Game Over state
Main Success Scenario:	1. The Player Avatar gets hit by an enemy attack 2. The Player Avatar Health decreases based on the damage value of the enemy attack 3. Player Avatar changes to an immovable state 4. The Player Avatar changes into a death animation 5. The player state changes to a Game Over state 6. Use Case ends
Extensions:	

UC ID and Name:	UC-9: Player Triggers Menu Option
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	
Trigger:	The player presses the Left-Mouse Button on a menu element
Description:	The player wants to interact with a menu element
Preconditions:	PRE-1. Player is in a Pause, Game Over, or Game Menu state

Postconditions:	POST-1. Menu option selection changes menu or game state
Main Success Scenario:	1. The player highlights a menu option 2. The player presses the Left-Mouse Button over the menu option 3. Menu option runs 4. Use Case ends
Extensions:	**4a. Restart Level** 4a1. Game state for level is reset to parameters on original entry 4a2. Use Case ends **4b. Quit Game** 4b1. Game executable is closed 4b2. Use Case ends **4c. Quit to map** 4c1. Current screen changes to map screen 4c2. Use Case ends **4d. Play Game** 4d1. Game progresses to level selection menu 4d2. Use Case ends

UC ID and Name:	UC-10: Player Pauses Game
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	
Trigger:	The player presses the ESCAPE key
Description:	The player wants to pause the game
Preconditions:	PRE-1. Player is in an Active Game state
Postconditions:	POST-1. Gameplay stops POST-2. Pause Menu options appear

	POST-3. Gameplay visually fades
Main Success Scenario:	1. The Player presses the ESCAPE key 2. Pause Menu appears 3. Gameplay fades to the background 4. Use Case ends
Extensions:	

UC ID and Name:	UC-11: Player Unpauses Game
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	
Trigger:	The player presses the ESCAPE key
Description:	The player wants to unpause the game
Preconditions:	PRE-1. Player is in a Pause state
Postconditions:	POST-1. Gameplay unfades POST-2. Pause Menu options disappear POST-3. Gameplay resumes
Main Success Scenario:	1. The Player presses the ESCAPE key 2. Gameplay unfades from the background 3. Pause Menu disappears 4. Use Case ends
Extensions:	

UC ID and Name:	UC-12: Mid-Combat Event Occurs
Created By:	
Date Created:	
Primary Actor:	Boss Enemy
Secondary Actors:	Player Avatar

Trigger:	Boss Enemy Reaches a Health Point threshold
Description:	The Boss Enemy uses a special attack, the effect of which changes depending on the enemy
Preconditions:	PRE-1. Boss Enemy Health is below threshold PRE-2. Boss Enemy is not currently performing a different attack
Postconditions:	
Main Success Scenario:	1. The Player Avatar attacks the enemy 2. Enemy health reduces below threshold value 3. Mid combat event occurs 4. Player interacts with event 5. Use Case ends
Extensions:	**5a. Enemy Dies During Mid Combat Event** 5a1. GO TO UC-14: Enemy Dies

UC ID and Name:	UC-13: Enemy Takes Damage
Created By:	
Date Created:	
Primary Actor:	Player Avatar
Secondary Actors:	Enemy
Trigger:	Enemy is hit by Player Avatar attack or damaging spell
Description:	The enemy takes damage from a Player Avatar attack
Preconditions:	PRE-1. Enemy health is above 0 PRE-2. Enemy is not currently in an invulnerable state
Postconditions:	POST-1. Enemy sprite flashes POST-2. Enemy Health bar updates based on new value
Main Success Scenario:	1. The enemy gets hit by a player attack 2. The enemy Health decreases based on the damage value of the Player Avatar attack 3. The enemy sprite flashes 4. Use Case ends
Extensions:	**5a. Enemy Dies**

	5a1. GO TO UC-14: Enemy Dies

UC ID and Name:	UC-14: Enemy Dies
Created By:	
Date Created:	
Primary Actor:	Enemy
Secondary Actors:	Player Avatar
Trigger:	Enemy is hit by Player Avatar attack or damaging spell
Description:	The enemy takes damage from a Player Avatar attack and dies
Preconditions:	PRE-1. Enemy health is above 0 PRE-2. Enemy is not currently in an invulnerable state
Postconditions:	POST-1. Enemy sprite flashes POST-2. Enemy Health bar updates based on new value POST-3. Enemy death animation plays
Main Success Scenario:	1. The enemy gets hit by a Player Avatar attack 2. The enemy health decreases based on the damage value of the player attack 3. The enemy sprite flashes 4. Enemy health is reduced to 0 5. Enemy death animation plays 6. Use Case ends
Extensions:	

UC ID and Name:	UC-15: Enemy Attacks
Created By:	
Date Created:	
Primary Actor:	Enemy
Secondary Actors:	Player Avatar
Trigger:	Enemy Action Cooldown reached 0
Description:	The enemy performs an attack

Preconditions:	PRE-1. Enemy health is above 0
Postconditions:	POST-1. Enemy uses an attack POST-2. Enemy Action Cooldown starts
Main Success Scenario:	1. The enemy uses an attack 2. The enemy Action Cooldown starts 3. Use Case ends
Extensions:	

UC ID and Name:	UC-16: Boss Enemy Intro Cutscene
Created By:	
Date Created:	
Primary Actor:	Player Avatar
Secondary Actors:	Boss Enemy
Trigger:	Player enters a level
Description:	The Player Avatar and Boss Enemy perform introductory animations to set the context of the scene
Preconditions:	PRE-1. Player Avatar is in an immovable state PRE-2. Boss Enemy attack logic is paused
Postconditions:	POST-1. Player Avatar is an a moveable state POST-2. Boss Enemy attack logic begins
Main Success Scenario:	1. The player selects a level 2. The Player Avatar and the Boss Enemy perform entrance animations 3. Use Case ends
Extensions:	**3a. Game Tutorial** 3a1. The Player Avatar is able to be controlled 3a2. GOTO UC-17

UC ID and Name:	UC-17: Game Tutorial

Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Boss Enemy, Player Avatar
Trigger:	Player enters the first level
Description:	The player is given a tutorial on how to control the Player Avatar
Preconditions:	PRE-1. Player Avatar is in an moveable state PRE-2. Boss Enemy attack logic is paused
Postconditions:	POST-1. Boss Enemy attack logic begins
Main Success Scenario:	1. The first tutorial message plays 2. The player follows the instructions of the first tutorial message and controls the Player Avatar accordingly 3. The second tutorial message plays 4. The player follows the instructions of the second tutorial message and controls the Player Avatar accordingly 5. The third tutorial message plays 6. The player follows the instructions of the third tutorial message and controls the Player Avatar accordingly 7. The fourth tutorial message plays 8. Use Case ends
Extensions:	**Xa. Exit Tutorial Early** Xa1. The Player Avatar damages the Boss Enemy Xa2. Use Case ends

UC ID and Name:	UC-18: Player Avatar Swap Spells
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	Player pressed 1, 2, or 3 Key
Description:	The player swaps the Player Avatar's currently active spell

Preconditions:	PRE-1. Player Avatar is in an moveable state
Postconditions:	POST-1. Player Avatar active spell has changed
Main Success Scenario:	1. The player presses the 1, 2, or 3 key 2. Player Avatar active spell changes to the respectively pressed button 3. An animation plays above the Player Avatar showing the newly selected spell 4. The spell UI highlights the newly active spell 5. Use Case ends
Extensions:	

UC ID and Name:	UC-19: Boss Fight Ending Cutscene
Created By:	
Date Created:	
Primary Actor:	Player Avatar
Secondary Actors:	Boss Enemy
Trigger:	Player has defeated Boss Enemy
Description:	An ending cutscene plays to resolve the conflict of the Boss Fight
Preconditions:	PRE-1. Boss Enemy is dead
Postconditions:	POST-1. Boss Enemy level is disabled in level menu
Main Success Scenario:	1. The Boss Enemy plays its death animation 2. Player Avatar positions self to appropriate location for cutscene 3. A cutscene plays 4. Player Avatar plays a level exit animation 5. Use Case ends
Extensions:	

UC ID and Name:	UC-20: Celestial Primate Lightning Bolt
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	Player Avatar
Trigger:	Celestial Primate Action Cooldown is at 0, Attack has been selected randomly from attacks available in current phase
Description:	The Celestial Primate attacks using a volley of lightning bolts
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. UC-15 is being extended PRE-4. Current phase includes this attack
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate telegraphs a lightning bolt attack at the Player Avatar's current position 2. Lightning bolt fires at the end of its telegraph duration 3. Simultaneously, the Celestial Primate continues to telegraph 3 more lightning bolt attacks with a short delay between them 4. Use Case ends
Extensions:	

UC ID and Name:	UC-21: Celestial Primate Meteor Swarm
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	Player Avatar
Trigger:	Celestial Primate Action Cooldown is at 0, Attack has been selected randomly from attacks available in current phase
Description:	The Celestial Primate attacks using a swarm of tracking meteors
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. UC-15 is being extended PRE-4. Current phase includes this attack

Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate shoots a swarm of small meteors up into the air 2. The meteors track the Player Avatar's location for a short duration 3. The meteors continue on their current trajectory when the tracking duration has elapsed 4. Use Case ends
Extensions:	

UC ID and Name:	UC-22: Celestial Primate Ice Shock
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	Player Avatar
Trigger:	Celestial Primate Action Cooldown is at 0, Attack has been selected randomly from attacks available in current phase
Description:	The Celestial Primate chills a circular area that flash-freezes for damage at the end of the telegraph duration
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. UC-15 is being extended PRE-4. Current phase includes this attack
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate begins the ice shock attack at the Player Avatar's location 2. The ice shock animation plays 3. After the telegraph duration ends, the ice shock deals damage in the area 4. Use Case ends
Extensions:	**4a. Player Avatar hit** 4a1. Player Avatar gains a debuff that slows them for a short time

UC ID and Name:	UC-23: Celestial Primate Flame Pillars
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	Player Avatar
Trigger:	Celestial Primate Action Cooldown is at 0, Attack has been selected randomly from attacks available in current phase
Description:	The Celestial Primate attacks using columns of flame that appear simultaneously, equally spaced across the stage
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. UC-15 is being extended PRE-4. Current phase includes this attack
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate creates flame columns based on the Player Avatar's current position; the first column is placed at the Player Avatar's current location, and the other columns are created based on that initial column 2. The flame column telegraph animations play 3. After the telegraph duration ends, the flame columns deal damage in their respective areas 4. Use Case ends
Extensions:	

UC ID and Name:	UC-24: Celestial Primate Chaos Crush Mid-Combat Event
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	Player Avatar
Trigger:	Celestial Primate Action Cooldown is at 0, Phase 3 has begun

Description:	The Celestial Primate appears in the center of the stage and summons a black hole that tries to suck the Player Avatar in
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. UC-12 is being extended
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate teleports to the center of the stage 2. The Celestial Primate performs its Chaos Crush animation 3. A black hole is summoned underneath the Celestial Primate 4. A lotus seat is summoned beneath the black hole, at ground level 5. The Player Avatar is drawn into the black hole 6. Black hole closing animation runs 7. Use Case ends
Alternatives:	**5a. Lotus Seat Yin-Yang Defense** 5a1. Player Avatar stays motionless over lotus seat 5a2. An animation plays indicating the Player Avatar's safety 5a3. Use case continues at 6. **5b. Lotus Seat Yin-Yang Defense Fail** 5b1. Player Avatar stays motionless over lotus seat 5b2. An animation plays indicating the Player Avatar's safety 5b3. The Player Avatar moves 5b4. Use case continues at 6a.
Extensions:	**6a. Player Avatar gets sucked into black hole** 6a1. Player Avatar disappears for a short time before reappearing and taking damage 6a2. Use case continues at 6.

UC ID and Name:	UC-25: Enemy Movement
Created By:	
Date Created:	
Primary Actor:	Enemy

Secondary Actors:	
Trigger:	Enemy Action Cooldown is at 0, movement action has been selected
Description:	The Enemy moves to a different location
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. Enemy is alive
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Enemy selects a movement action 2. The Enemy performs the selected movement 3. Use Case ends
Extensions:	

UC ID and Name:	UC-26: Celestial Primate Straight Movement
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	
Trigger:	Celestial Primate Action Cooldown is at 0, movement action has been selected
Description:	The Celestial Primate moves between two Anchor Points in a straight line
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. Celestial Primate is alive PRE-4. UC-25 is in progress
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate selects the straight movement action 2. The Celestial Primate moves in a straight line between two Anchor Points 3. Use Case ends

Extensions:	

UC ID and Name:	UC-27: Celestial Primate Arc Movement
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	
Trigger:	Celestial Primate Action Cooldown is at 0, movement action has been selected
Description:	The Celestial Primate moves between two Anchor Points in an arc
Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. Celestial Primate is alive PRE-4. UC-25 is in progress
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate selects the arc movement action 2. The Celestial Primate moves in an arc between two Anchor Points 3. Use Case ends
Extensions:	

UC ID and Name:	UC-28: Celestial Primate Teleport Movement
Created By:	
Date Created:	
Primary Actor:	Celestial Primate
Secondary Actors:	
Trigger:	Celestial Primate Action Cooldown is at 0, movement action has been selected
Description:	The Celestial Primate teleports between two Anchor Points

Preconditions:	PRE-1. Player Avatar is alive PRE-2. Action Cooldown is 0 PRE-3. Celestial Primate is alive PRE-4. UC-25 is in progress
Postconditions:	POST-1. Action Cooldown is reset
Main Success Scenario:	1. The Celestial Primate selects the teleport movement action 2. The Celestial Primate plays a teleport-out animation 3. The Celestial Primate appears at the destination Anchor Point 4. The Celestial Primate plays a teleport-in animation 5. Use Case ends
Extensions:	

UC ID and Name:	UC-29: Boss Enemy Phase Change
Created By:	
Date Created:	
Primary Actor:	Boss Enemy
Secondary Actors:	Player Avatar
Trigger:	Boss Enemy Health reduces below phase threshold
Description:	The Boss Enemy changes phases, gaining new abilities and behaviour
Preconditions:	PRE-1. Boss Enemy is alive
Postconditions:	POST-1. Boss Enemy is in the next phase
Main Success Scenario:	1. The Boss Enemy loses Health that takes them below the phase change threshold 2. A visual effect plays to indicate a phase change has occurred 3. The Boss Enemy's phase moves to the next phase 4. Use Case ends
Extensions:	

UC ID and Name:	UC-30: Player Selects a Level
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	
Trigger:	The player presses the Left-Mouse Button on a level selection circle
Description:	The player wants to enter a level
Preconditions:	PRE-1. Player is in the Level Selection page
Postconditions:	POST-1. Selected level is loaded
Main Success Scenario:	1. The player highlights a level selection option 2. The player presses the Left-Mouse Button over the level 3. Level details menu opens and animates 4. Player selects level start button 5. Use Case ends
Alternatives:	**4a. Player Selects Different Level** 4a1. Newly selected level details menu opens and animates 4a2. Use Case continues at 4.
Extensions:	

UC ID and Name:	UC-31: Player Avatar Cast Fireball
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Right-Mouse Button when the Fireball spell is active
Description:	The player wants the Player Avatar to attack using a Fireball spell
Preconditions:	PRE-1. Player Avatar is in a moveable state PRE-2. Player Avatar's Mana is above the spell's Mana cost PRE-3. Currently active spell is Fireball PRE-4. UC-5 is currently active

	PRE-5. Fireball cooldown timer is at 0
Postconditions:	POST-1. The Player Avatar casts a spell POST-2. The Player Avatar plays a casting animation POST-3. A fireball spell effect is created POST-4. Fireball spell goes on cooldown for its cooldown duration
Main Success Scenario:	1. The Player presses the Right-Mouse Button 2. The Player Avatar casts a Fireball 3. The Player Avatar plays the Instant spellcast animation 4. Player Avatar Mana resource is reduced by the amount required by the fireball 5. Mana Resource bar is updated 6. Fireball moves horizontally in a straight line from where it was cast 7. Use Case ends
Extensions:	**7a. Fireball hit** 7a1. The fireball hits an enemy 7a2. Enemy takes damage value of fireball spell 7a3. Fireball impact animation plays 7a4. Use case ends

UC ID and Name:	UC-32: Player Avatar Cast Magic Explosion
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Right-Mouse Button when the Magic Explosion spell is active
Description:	The player wants the Player Avatar to attack using a Magic Explosion spell
Preconditions:	PRE-1. Player Avatar is in a moveable state PRE-2. Player Avatar's Mana is above the spell's Mana cost PRE-3. Currently active spell is Magic Explosion

	PRE-4. UC-5 is currently active
Postconditions:	POST-1. The Player Avatar casts a spell POST-2. The Player Avatar plays a casting animation POST-3. A Magic Explosion spell effect is created POST-4. Combo step increments
Main Success Scenario:	1. The Player presses the Right-Mouse Button 2. The Player Avatar casts a Magic Explosion 3. The Player Avatar plays the Magic Explosion spellcast animation for the current combo step 4. Player Avatar Mana resource is reduced by the amount required by the Magic Explosion's mana cost 5. Mana Resource bar is updated 6. Magic Explosion animation and effect plays 7. Use Case ends
Extensions:	**3a. Magic Explosion Combo Injection** 3a1. GOTO UC-5 3a

UC ID and Name:	UC-33: Player Avatar Cast Magic Missilerack
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player pushes the Right-Mouse Button when the Magic Missilerack spell is active
Description:	The player wants the Player Avatar to attack using a Magic Missilerack spell
Preconditions:	PRE-1. Player Avatar is in a moveable state PRE-2. Player Avatar's Mana is above the spell's Mana cost PRE-3. Currently active spell is Magic Missilerack PRE-4. UC-5 is currently active PRE-5. Magic Missilerack cooldown timer is at 0
Postconditions:	POST-1. The Player Avatar casts a spell POST-2. The Player Avatar plays a casting animation POST-3. A Magic Missilerack spell effect is created

	POST-4. Magic Missilerack goes on cooldown for its cooldown duration
Main Success Scenario:	1. The Player presses the Right-Mouse Button 2. The Player Avatar changes to the Incantation spellcasting animation 3. The Magic Missilerack fades in behind the Player Avatar while the cast time elapses 4. Cast time finishes 5. Player Avatar Mana resource is reduced by the amount required by the Magic Missilerack 6. Mana Resource bar is updated 7. Magic Missilerack shoots a volley of Magic Missiles 8. Magic Missiles track the location of an Enemy until it hits them 9. Enemy hit by Magic Missile take damage value of that spell 10. Use Case ends
Extensions:	**4a. Player Avatar interrupted by hit** 4a1. The enemy hits the Player Avatar 4a2. Player Avatar casting is interrupted and Player Avatar state is reset to idle 4a3. GOTO UC-7

UC ID and Name:	UC-34: Player Avatar Falling Attack
Created By:	
Date Created:	
Primary Actor:	Player
Secondary Actors:	Player Avatar
Trigger:	The player attacks or casts a spell while the Player Avatar is in a falling state
Description:	The player wants the Player Avatar to attack while falling in the air
Preconditions:	PRE-1. Player Avatar is in a falling state PRE-2. Player Avatar is in a moveable state
Postconditions:	POST-1. The Player Avatar attacks in the air

	POST-2. The Player Avatar's falling speed is reduced for the duration of the attack
Main Success Scenario:	1. The Player presses the LMB 2. The Player Avatar attacks as described in UC-3 3. The Player Avatar's falling speed is reduced 4. UC-3 finishes 5. The Player Avatar falling speed reverts to default 6. The Player Avatar touches the ground 7. Use Case ends
Alternatives:	**1a. Mid-air Combo Spell** PRE-3. Currently active spell is a Combo spell 1a1. The Player presses the RMB 1a2. The Player Avatar casts a Combo spell, as described in UC-5, taking the 3a extension 1a3. The Player Avatar's falling speed is reduced 1a4. UC-5 finishes 1a5. The Player Avatar falling speed reverts to default 1a6. The Player Avatar touches the ground 1a7. Use Case ends **1b. Mid-air Incantation Spell** PRE-3. Currently active spell is an Incantation spell 1b1. The Player presses the RMB 1b2. The Player Avatar stops falling 1b3. UC-5 proceeds, taking the 3b extension 1b4. UC-5 finishes 1b5. Player Avatar starts falling again 1b6. Use Case ends
Extensions:	