**AI Assisted Game Development**

*Developing a 3D Game While Incorporating AI*



**Wicked West**

Megan Carver, Kyle Nepo, Nathanael Ostheller, John Kendall

**TABLE OF CONTENTS**

[Project Introduction 5](#_Toc152856729)

[Background and Related Work 5](#_Toc152856730)

[Project Overview 6](#_Toc152856731)

[Client and Stakeholder Identification and Preferences 8](#_Toc152856732)

[Team Members – Bios and Project Roles 9](#_Toc152856733)

[System Requirements Specification 11](#_Toc152856734)

[Use Cases 11](#_Toc152856735)

[Functional Requirements 15](#_Toc152856736)

[1. Generative AI 15](#_Toc152856737)

[2. Core Game Systems 16](#_Toc152856738)

[3. Unity Engine Implementation 17](#_Toc152856739)

[Non-Functional Requirements 17](#_Toc152856740)

[System Evolution 18](#_Toc152856741)

[System Overview 20](#_Toc152856742)

[Architecture Design 20](#_Toc152856743)

[Overview 20](#_Toc152856744)

[Subsystem Decomposition 21](#_Toc152856745)

[1. [UI Handler] 21](#_Toc152856746)

[1.1. Description 21](#_Toc152856747)

[1.2. Concepts and Algorithms Generated 21](#_Toc152856748)

[1.3. Interface Description 21](#_Toc152856749)

[2. [Game Environment] 22](#_Toc152856750)

[2.1. Description 22](#_Toc152856751)

[2.2. Concepts and Algorithms Generated 22](#_Toc152856752)

[2.3. Interface Description 22](#_Toc152856753)

[3. [Input Handler] 22](#_Toc152856754)

[3.1. Description 22](#_Toc152856755)

[3.2. Concepts and Algorithms Generated 22](#_Toc152856756)

[3.3. Interface Description 22](#_Toc152856757)

[4. [World Loader] 23](#_Toc152856758)

[4.1. Description 23](#_Toc152856759)

[4.2. Concepts and Algorithms Generated 23](#_Toc152856760)

[4.3. Interface Description 23](#_Toc152856761)

[5. [Dialogue System] 23](#_Toc152856762)

[5.1. Description 23](#_Toc152856763)

[5.2. Concepts and Algorithms Generated 23](#_Toc152856764)

[5.3. Interface Description 23](#_Toc152856765)

[6. [ChatGPT Server Handler] 24](#_Toc152856766)

[6.1. Description 24](#_Toc152856767)

[6.2. Concepts and Algorithms Generated 24](#_Toc152856768)

[6.3. Interface Description 24](#_Toc152856769)

[7. [Prompt State Handler] 24](#_Toc152856770)

[7.1. Description 24](#_Toc152856771)

[7.2. Concepts and Algorithms Generated 24](#_Toc152856772)

[7.3. Interface Description 24](#_Toc152856773)

[8. [Game State Handler] 25](#_Toc152856774)

[8.1. Description 25](#_Toc152856775)

[8.2. Concepts and Algorithms Generated 25](#_Toc152856776)

[8.3. Interface Description 25](#_Toc152856777)

[Data design 25](#_Toc152856778)

[User Interface Design 26](#_Toc152856779)

[Testing Introduction 27](#_Toc152856780)

[Project Overview 27](#_Toc152856781)

[Test Objectives and Schedule 27](#_Toc152856782)

[Testing Strategy 27](#_Toc152856783)

[Test Plans 28](#_Toc152856784)

[Unit Testing 28](#_Toc152856785)

[Integration Testing 28](#_Toc152856786)

[System Testing 29](#_Toc152856787)

[Functional testing: 29](#_Toc152856788)

[Performance testing: 29](#_Toc152856789)

[User Acceptance Testing: 29](#_Toc152856790)

[Environment Requirements 30](#_Toc152856791)

[Inventory System Expansion 32](#_Toc152856792)

[Saving and Loading Updates 32](#_Toc152856793)

[Event Tracking 32](#_Toc152856794)

[World Updates 32](#_Toc152856795)

[NPCs 32](#_Toc152856796)

[Hiding Action 32](#_Toc152856797)

[The Monster 32](#_Toc152856798)

[Journal 32](#_Toc152856799)

[I.1. UI Handler 35](#_Toc152856800)

[I.1.1. Functions and Interfaces Implemented 35](#_Toc152856801)

[I.1.2. Preliminary Tests 35](#_Toc152856802)

[I.2. Game Environment 36](#_Toc152856803)

[I.2.1. Functions and Interfaces Implemented 36](#_Toc152856804)

[I.2.2. Preliminary Tests 37](#_Toc152856805)

[I.3. Input Handler 37](#_Toc152856806)

[I.3.1. Functions and Interfaces Implemented 37](#_Toc152856807)

[I.4. World Loader and Game State Handler 37](#_Toc152856808)

[I.4.1. Functions and Interfaces Implemented 37](#_Toc152856809)

[I.4.2. Preliminary Tests 37](#_Toc152856810)

[I.5. Dialogue System, Prompt State Handler, ChatGPT Server Handler 39](#_Toc152856811)

[I.5.1. Functions and Interfaces Implemented 39](#_Toc152856812)

[I.5.2. Preliminary Tests 39](#_Toc152856813)

[Glossary 42](#_Toc152856814)

[Appendices 43](#_Toc152856815)

[References 48](#_Toc152856816)

*Introduction*

# Project Introduction

Our group is tasked with producing a demo game created with help from ChatGPT or Bard for Microsoft’s Mojang Studios Division. The final deliverable comes in the form of a publishable game. Our group has decided on making a 3D Western Fantasy game which incorporates Chat GPT interactions with a cast of NPCs in a small mining town.

The motivation behind our project comes from both our shared desire to explore and create projects which blend AI and game development, and our shared desire to complete a publishable game in our Capstone timeframe. The idea for specifically a 3D Western Fantasy game comes from the combination of each members’ individual skillsets. Megan Carver has experience and excels in 3D modeling and design, while Kyle Nepo and Nathanael Ostheller both have a history in making games, creating storyboards, and experience in the phase of game design choices. Their experience has been shown in the ideas they have already come up with for the chosen plot of our game. John Kendall has experience in connecting OpenAI’s application programming interface (API) to remote Ubuntu servers for independent Unity games, making Unity the game engine of choice for developing a well-received game as the project outcome specifies [5] [1].

The process of incorporating AI in development, along with the deliverance of the project itself, will be used to help aid the Mojang Studios division under Microsoft in research on combining AI with game development.

# Background and Related Work

To evaluate the current industry leaders in our project field, it becomes necessary to fully define what exactly is considered the domain of our project. For this document, we are specifically researching “*AI legally used in the development of available games*”. Many different entertainment companies have reported beginnings of research on using generative AI to develop whatever project is being worked on in their studio, however, much of this is being kept very secretive and little information is present beyond that the studios are using it. There are still some projects that have been published that use generative AI in its development, but many of these projects are closer to a tech demo than a fully published game. Our research highlights three of these industry leading projects that have information about them publicly available.

The first notable example is the recently released technical demo Inworld Origins [3]. This playable demo was created by Inworld AI and released on Steam July 23, 2023. The free to play demo is particularly notable in the fact that its ensemble cast of NPCs are controlled by the in-house developed Inworld AI system. This system is the most similar of the cases that will be analyzed for the purposes of this project. Inworld AI’s system allows developers to give the AI system parameters about the character’s personality, relationships, and context for the story and world. From this, Inworld’s system generates dynamic text responding to the inputs given to the NPC by the user. While in concept this engine sounds very promising, in practice the tech demo received mixed responses from players. The demo was reportedly filled with bugs, and it also had issues with the NPC’s veering wildly off course and becoming completely removed from the world and story the game was taking place in. With our development of the Wicked West project, we hope for our prompting and generative AI system to prevent the generative AI responses from veering away from the world that has been set and prevent tangents that detract from the immersive experience that these improved NPCs hope to provide.

The next example, while not credited in any finished and released games, is an AI tool being developed by a notable game creator and publisher that has had information made available to the public through written press releases from the publisher. This item is the in-house AI tool created by La Forge, Ubisoft’s R&D Department, Ghostwriter [6]. Ghostwriter was created as a supplemental tool for video game writers and is particularly used for writing drafts of barks. This allows writers to spend more time working on more important areas of story narrative and dialogue, while still having relevant barks in the game world to help immerse the player and make the world feel alive. These draft barks delivered by Ghostwriter are edited and adjusted by writers if needed after delivered by the AI. The main difference between this tool and what our group is attempting to do with the Wicked West project is that the dialogue developed by Ghostwriter is generated in the writing room and edited by writers before it is seen anywhere near a finished project. With the Wicked West project, the dialogue we will be using from generative AI will be generated dynamically as the game runs and goes directly from the generative AI model to the player without any developer intercedence. This means that what is created for our project must be very finely tuned, as we do not have the opportunity to adjust and fix it after it is created.

The final example from the research done for this project is SQUARE ENIX’s AI tech preview The Portopia Serial Murder Case [7]. This piece of software is a demonstration of Natural Language Processing as applied to the Japanese published text adventure game of the same name that was written in 1983. This software keeps the original story and events from the first publishing of The Portopia Serial Murder Case but updates the Natural Language Processing to modern levels allowing greater freedom in the experience of the story. The in-house draft of the project also included Natural Language Generation, where the system would generate responses to questions that did not have one pre-written. This was omitted from the released product due to the often occurrence of AI generating “unethical responses”. This tech demo was poorly received by modern players, as the removal of the Natural Language Generation made the game feel unresponsive and difficult to play as the written responses from the 80s did not cover enough of the common options players tried to take. This project only uses half of the technologies we expect to use with the Wicked West project, as we plan to use Natural Language Processing and Generation in our system of communication with NPCs.

The main technical skills that our group will need to learn for this project will be the integration of the generative AI into the dialogue system we will be using in Unity, as well as the skill in prompting and adjusting prompts to ensure that the dialogue that is created is in character and furthers the plot as we want it to. We will also likely need to learn new technical skills and techniques that are prevalent in game development and the Unity Engine, as there are sure to be pieces of the game’s development that we do not have a defined solution ready for.

# Project Overview

AI has potential in many different fields, and game development is no exception. Our game, *Wicked West*, aspires to prove AI as a useful tool in game development. To do so, we plan to use AI for NPC interactions in the game world. It will make use of ChatGPT's model by writing prompts, allowing real time updating of character dialogue and decisions. The finished product using these systems would help to prove AI's value as a tool in game development.

To reach this goal, our project team will need to complete many objectives. Our overall goal is to create an enjoyable game, as doing so will demonstrate the capabilities of AI and deliver a final product that our team can be proud to have produced. To achieve this, we decided to create a mystery game with a story similar to games such as *Mafia* or *Town of Salem*, in which the player would try to discover the culprit in a town where people go missing. To accomplish this, we aim to create 8 individual NPCs (Non-Playable Characters) who will interact with the player character, as well as one main antagonist. The game will allow the player to travel through a small town over the course of a week, finding clues and items that will help them deduce the truth.

Because this project relies on it, it is important that we define the ways in which AI will be used. As such, our primary use of AI in our game will be in character dialogue. As ChatGPT's model for writing AI changes depending on the prompts used, we will provide prompts detailing a character’s background and way of speech. The main feature of the prompts will be the character's knowledge of the other NPCs in the game and events that have transpired in the game world. Our antagonist will have a special ability to erase others from existence, and we will replicate this by erasing the knowledge of removed characters from other NPC’s prompts. The player will be able to ask questions by typing dialogue which is answered in real time. We will have to decide whether the prompt is reset after every few questions to keep the NPC focused or if we let the player continue to take conversations off-topic; whichever provides the most enjoyable experience. This way, we will be able to utilize AI in a creative way that further enhances the game experience. On an additional note, we may also use AI to assist us in programming. We hope that the use of AI in programming will help decrease time debugging, allowing us more time on design decisions and artwork.

Our project has clear lines drawn in which AI will be used. While we will be using AI to write dialogue for characters to utilize it for its real time capabilities, we will not be using AI for other creative aspects such as art and game design. Though AI would be able to provide such, we do not believe it necessary in proving its capabilities in the field of game development.

For further specifications, we will be designing this game in Unity, using its 3D engine, and building the game solely for PC. It is important that our game remains optimized while using an AI language model in real-time. Our team is responsible for modeling and texturing many of the in-game assets, but we may use assets from the Unity store or other sources with assets that are allowed for use in published games to help cut down on production time. Furthermore, our open world will allow players to enter primary buildings such as the town hall, but others may be closed off and only used to decorate the game world. Our world will be medium size, with about 5 buildings able to be entered and an explorable outdoor environment. Our NPCs will likely be stationary but be able to shift around the town depending on game time.

*Wicked West* is a single-player Player vs Environment experience, with a faceless main character allowing the player to step into their shoes. The player's main goal will be to find a way to defeat the monster within a set deadline. To do so, they will have to explore the town and question the NPC's. Interactions will utilize a dialogue box that prompts for player dialogue as well as provides art of the NPC being spoken to. Each NPC will have their own personality and job. Various clues may be picked up and interacted with as a way of furthering the player towards the goal of defeating the monster. *Wicked West* will also be based in a western-fantasy setting, and so our art assets match that design.

Our final product will be a game we can publish on a platform such as Steam or itch.io. Depending on the service model used, we may only be able to utilize its server connection for a limited time. Our team hopes to provide a fun, enjoyable experience that helps showcase the capabilities of AI in game development.

# Client and Stakeholder Identification and Preferences

Our primary client is Microsoft. However, our point of contact at Microsoft is no longer available, so Professor Ananth Jillepalli will be serving as our client until we are able to secure a new contact at Microsoft. We are working on this project for Mojang Studios, so they are our primary client as well, but since Microsoft owns Mojang studios, we will use their names interchangeably.

To explore the extent to which AI can be used for game development, we are creating a 3D game that uses Chat GPT to play eight different non-playable characters. We will likely be using AI to aid in the development process of our game as well. Our final deliverable will be a fully functional 3D game that we are hoping to publish.

Potential clients could also include any future players of our game. To appeal to these clients, our game must be challenging enough to keep the player occupied and entertained, and it should take time for a player to complete all stages of the game. The game environment should also appeal visually to the player.

In summary, incorporating AI into the development of our game will aid Microsoft and Mojang in research regarding AI and game development. The players of our game will benefit from challenging levels and visually appealing graphics. Our team will work to treat these needs as we build our game. The needs of our primary client, Microsoft, or Professor Ananth Jillepalli until we have a new contact at Microsoft, will be prioritized first, but the needs of the players will be considered throughout the design and development process.

*Team*

# Team Members – Bios and Project Roles

**Megan Carver**

Megan Carver is a computer science student interested in 3D modeling and rigging for video games and animated films. She currently works for Apple as a project manager for the development of a sales training program and has prior experience as a teaching assistant for the 3D animation class at WSU. Megan’s skills include, Java, Python, C++, HTML, CSS, Swift, 3D modeling, 3D animation, materials, rigging, and project management. Megan served as the team lead and the main 3D artist for this project. Her responsibilities include creating the 3D models and materials for the game, including but not limited to all the buildings and the building interiors in the game. She was also responsible for importing all of the models into Unity, adding interactive elements to the models, and mapping the game environment.

**John Kendall**

John Kendall is a software engineer who has a background in game development since childhood. He has a background in competition in online game development competitions, and experience in making small-scale indie games with Flash, Java, Unity, and more. John has experience as a teaching assistant for WSU’s CPT\_S 302 Ethics in Computation course, and experience in database systems, integrating game development platforms with the web/API, programming in C/C++, C#, Java, Javascript, Python, HTML and CSS, SQL, basic website design, firewalling, setting up remote Ubuntu severs, and experience in integrating systems such as Unity and websites with OpenAI’s ChatGPT models. John served as a programmer and managed the outside systems integrated into the project, such as OpenAI ChatGPT Unity packages, API key management, and server setup. His responsibilities include updating the team on API usage through the development process, integrating packages for Unity with their respective API needs, stepping in on Unity whenever help is needed by others, and ensuring that the overall Unity project is programmed in an optimal way. In terms of specifics relating to the game deliverable, his responsibilities include, but are not limited to, saving/loading functionality, pause menu functionalities, GUI functionalities, creating and maintaining pausing features, dialogue system integration, API setup, and maintaining a high-level of documentation.

**Nathanael Ostheller**

Nathanael Ostheller is a computer science student with a certification in game design and an interest in game development and game programming. He has been a part of some game development projects in the past, with development of some of those projects still being ongoing. Nathanael’s skills include object-oriented programming in C#, C++, and Java, experience with game engines such as Unity and Godot, and game design and writing. Nathanael’s roles have been as one of the gameplay programmers, as well as a writer and designer for the overall game. His responsibilities include the implementation of the player character and several of the systems surrounding it, such as the inventory, as well as conceptual design, plot writing, and character writing.

**Kyle Nepo**

Kyle Nepo is a computer science undergraduate with a history in game development. In past projects, he had much experience in level design and character design. He also has experience with 3D modeling, animation, and rigging, working on many projects for games and animations. For past projects, Kyle has used C#, Java, and Python, as well as other game engines with their own languages. He has also worked on many teams in past projects and learned effective project management. Due to his background in 2D art and design, Kyle will be the main artist for the game and its assets. His responsibilities include the art design of items, characters, and menu assets, providing 2D images and 3D models, as well as any programming required for in-game systems.

*Project Requirements*

# System Requirements Specification

In this section, we describe the features, functions, and other specifications that are requirements for our product.

## Use Cases

A diagram of a application

Description automatically generated

|  |  |
| --- | --- |
| **Start New Game** | |
| **Pre-condition** | The user is on the start game screen |
| **Post-condition** | - game information and instructions are displayed  - the game environment is loaded |
| **Basic Path** | 1. Locate ’start new game’ on the start game screen  2. Click the ‘Start New Game’ button |
| **Alternative Path** | - While playing the game, the user pauses the game to pull up the game menu. The player selects the ‘Exit to Menu’ option to return to the start menu and then does step 1 and step 2  - At step 2, instead of clicking the button, the user uses the up and down arrow keys or the w and s keys to select the ‘start new game button, then presses the Enter key to select |
| **Related Requirements** | - Pause game  - Load saved game |

|  |  |
| --- | --- |
| **Load Saved Game** | |
| **Pre-condition** | The user is on the start game screen |
| **Post-condition** | - game information is displayed  - the game environment is loaded |
| **Basic Path** | 1. Locate ’Load Saved Game’ on the start game screen  2. Click the ‘Load Saved Game’ button |
| **Alternative Path** | 1. The user uses the up and down arrow keys or the w and s keys to select the ‘Load Saved Game’ button  2. The user presses the ‘Enter’ key to select |
| **Related Requirements** | - Start new game |

|  |  |
| --- | --- |
| **Pause Game** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | The game pauses and the menu appears |
| **Basic Path** | 1. The user presses the Esc key  2. The user selects the button to ‘Exit to Desktop’ or ‘Exit to Menu’ to save and exit |
| **Alternative Path** | 2. The user selects ‘Resume’ to return to their game |
| **Related Requirements** | - View Game Manual |

|  |  |
| --- | --- |
| **View Game Manual** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | - Game manual is displayed |
| **Basic Path** | 1. Press the ‘Esc’ key to pause the game and pull up the menu  2. Locate the ‘Journal’ option  3. Click on ‘Journal’ to view game instructions |
| **Alternative Path** | 2. The user uses the arrow keys or the AWSD keys to locate the ‘Journal’ option  3. The user presses the ‘Enter’ key to select |
| **Related Requirements** | - Pause Game |

|  |  |
| --- | --- |
| **Start Conversation with NPC** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | The user enters a chat window with the NPC |
| **Basic Path** | 1. Walk in range of an NPC  2. Press the E or F key to interact |
| **Alternative Path** |  |
| **Related Requirements** | - Pick up objects  - Open doors  - Walk around |

|  |  |
| --- | --- |
| **Exit Conversation with NPC** | |
| **Pre-condition** | The user is in a conversation with an NPC |
| **Post-condition** | The user returns back to the game environment |
| **Basic Path** | 1. Press the ‘Esc’ key to exit the conversation |
| **Alternative Path** |  |
| **Related Requirements** | - Start conversation with NPC |

|  |  |
| --- | --- |
| **Pick Up Objects** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | The player character is holding an object from the game |
| **Basic Path** | 1. Walk up to the object  2. Press the E or F keys to pick the object up  3. Press the E or F key twice to release the object |
| **Alternative Path** |  |
| **Related Requirements** | - Start conversation with NPC |

|  |  |
| --- | --- |
| **Add Item to Inventory** | |
| **Pre-condition** | The player is in range of an item that can be picked up |
| **Post-condition** | The item has been added to the player’s inventory |
| **Basic Path** | 1. The player approaches an item  2. Press the E or F keys to interact with the object  3. The item is added to the players inventory upon interaction |
| **Alternative Path** |  |
| **Related Requirements** | - Pick up objects |

|  |  |
| --- | --- |
| **View Inventory** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | The inventory menu is displayed |
| **Basic Path** | 1. The user presses the “i” key  2. The user presses the I key again to return to the game. |
| **Alternative Path** |  |
| **Related Requirements** | - Add item to inventory |

|  |  |
| --- | --- |
| **Move Around** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | The user’s location in the game environment changes |
| **Basic Path** | 1. Hold down the A key to move left, the D key to move right, the W key to move forward, and the S key to move backwards  2. Hold down the shift key while using AWSD to run |
| **Alternative Path** | The user uses arrow keys instead if the AWSD keys to dictate direction |
| **Related Requirements** | - Pick up objects  - Open doors  - Start conversation with NPC |

|  |  |
| --- | --- |
| **Open Doors** | |
| **Pre-condition** | The user is actively playing the game and is not on a menu screen |
| **Post-condition** | The door opens and the user is able to walk through the doorway |
| **Basic Path** | 1. Walk up to the door  2. The door will automatically open when the player is close enough to it  3. The door will close automatically behind the player after the player leaves the area of the door |
| **Alternative Path** |  |
| **Related Requirements** | - Walk around |

## Functional Requirements

## Generative AI

**AI Integration**

|  |  |
| --- | --- |
| **Description** | The application must be able to utilize AI to generate dialogue for NPCs in the game. It needs to be able to respond top layer questions in real time and provide sufficient responses. It must also be able to update its own prompts. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 0: Essential and required functionality. |

## Core Game Systems

**Open World**

|  |  |
| --- | --- |
| **Description** | The game must have an open world with a minimum of 5 building players can enter. Buildings should be interactable, with some only able to be entered during certain in-game events. NPCs should be placed around the world according to in-game schedules. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 1: Desired functionality. |

**Non-Playable Characters**

|  |  |
| --- | --- |
| **Description** | The game must have a minimum of 8 non-playable characters the player can interact with. These NPC’s must utilize a generative language model to respond to player questions. They must also be unique and distinct through talking patterns and design. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 0: Essential and required functionality. |

**Interactive Items**

|  |  |
| --- | --- |
| **Description** | The game must include items that can be used by the player as clues and “investigation” items. They should be hidden around the map and spawned in during in-game events. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 1: Desired functionality. |

**In-Game Schedule**

|  |  |
| --- | --- |
| **Description** | The game must include items that can be used by the player as clues and “investigation” items. They should be hidden around the map and spawned in during in-game events. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 1: Desired functionality. |

**Save and Load Functions**

|  |  |
| --- | --- |
| **Description** | The game must include a save and load function that allows their game progress to be stored for later play. This should include stats such as gametime and prompts for the AI. This will be stored locally. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 2: Extra features or stretch goals. |

**Win/Lose Condition**

|  |  |
| --- | --- |
| **Description** | The game must include a win condition that is triggered by the player defeating the main monster of the game. It must also contain a lose condition caused by the player failing to defeat the monster. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 0: Desired functionality. |

## Unity Engine Implementation

**Open World**

|  |  |
| --- | --- |
| **Description** | The game must be run on the Unity engine. It must be able to use assets from the Maya 3D Modeling editor. It may also use assets from the Unity Asset store. The finished project will then be published. |
| **Source** | Requirements provided by members of the team. |
| **Priority** | Priority Level 0: Desired functionality. |

## Non-Functional Requirements

**Completeness:**

The story content and overall plot of the game should be self-contained, telling a complete story that does not require outside knowledge of the setting to understand and that concludes by the completion of play.

**Enjoyable:**

The overall game should be fun for users to play and should provide entertainment value to players.

**Stylistically Uniform:**

The menus, crosshair, and any other graphic user interfaces seen in the game should conform to the established game theme. This theme is intended to fall somewhere between low fantasy and western.

**Responsive:**

The dialogue and NPC reactions in the game should change to fit how the player interacts with NPCs and allow for multiple different play styles and ways of interacting with NPCs.

**Efficient Performance:**

The game should perform efficiently enough to be playable on modern midrange processors and lower end graphic cards or inbuilt system graphics. Playable in this case means that the game runs at or above 30 frames per second without noticeable stuttering or slowdowns during standard gameplay.

**Visually Intuitive:**

The graphic user interface should provide visual feedback on availability of actions and allow for clear navigation and understanding of what actions the player character can perform or is performing.

**Development Process:**

The game shall be developed using Unity Engine and C# code. Elements can be developed in other programs if the final product of the element can be exported to be used in Unity.

# System Evolution

Our project needs to be compatible with OpenAI applications for years to come, which comes with OpenAI’s requirement for identification and authentication of the user. There is really no way to hide API keys in the front-end code, so attempts at obfuscation and encryption of keys used to connect our project to OpenAI’s servers must be made through hiding our OpenAI private API keys behind a server which takes proxy requests to obtain the needed private keys. Our project’s largest system evolution decision is based around the need of a back-end proxy server to hide keys, since all other data is to be stored locally through static or serializable methods to not overcomplicate things. The server will be set up through DigitalOcean’s cloud server services. A DigitalOcean Droplet will hold an Ubuntu 22.04 (LTS) x64 server, of which the ipv4 will be used in connection with a Domain related to our game that is reachable by others and holds proper SSL Certification with DNS records. The DigitalOcean proxy server we will use is set up with a Premium second-generation Intel Xeon Scalable processor (1vCPU), 1GB of memory, 25GB NVMe SSD, and 1TB Transfer at $0.010/hr. As the game goes through development, resizing of the Droplet may need to occur if other data is desired to be added to the server rather than serialized in local files. This has been planned to be accounted for through possible switches to AMD if later desired, increases in Memory, and Transfer for sub-$0.021/hr. amounts. I do not believe that 2vCPUs will be needed at all but will be achievable through resizing of the droplet at any point in development or after if needed. Once fully developed, our proxy server can be switched out with a 512MB, 10GB SSD if that is found to be able to contain everything needed for hiding OpenAI API keys at $0.006/hr. This price could be subject to change at any time due to it being provided through DigitalOcean, so the possibility of that will have to be anticipated. Taking this option will be seen as a possibility near the end of production, since the size is not available to rescale our Droplet from due to 512MB Memory servers being run on a physically smaller disk size (regular Intel processor option would have to be shown to be compatible as well for this option to be picked). Any additional keys from additional AI sites could be hidden in this server as well, however our group has planned to keep data stored and transferred from the server as minimal as possible due to ongoing costs, server software evolution, and a desire to not overcomplicate our project.

When OpenAI suspends their gpt-3.5-turbo model without transferring API service requests elsewhere, then our project will no longer be able to function, in a way similar to Flash today. 3,500 requests-per-minute using Gpt-3.5-turbo service models will be acquired for our project via OpenAI API keys at a rate of 90,000 tokens-per-minute. Likewise, the rates of these could go up as OpenAI sees fit over time. This is something out of our control, although future suspensions of OpenAI GPT models be currently assumed to be very far in the future.

User needs will only change so much as systems develop overtime, since we are developing this game to be a locally-run, cloud-connectable fun, and overall enjoyable game to play that depends on Unity and Platform (Operating System) compatibility requirements. We can ensure that the game will be able to run on Windows, Mac, and Linux Standalone. In particular, Windows 7, 10, and 11 (64-bit only) will be able to run our game, however future versions of Windows or Operating Systems may not be compatible with our game over time. This can only be avoided by releasing the game compatible with several Operating Systems, thus our preparation to make the game compatible on more than just one system.

Client issues of any kind can come up, and the only thing in our control is to adapt when we can. Our client is currently Professor Ananth Jillepalli, so we must be ready to adapt at any stage in development to any future plans which could be desired from a Microsoft Capstone representative, if one is to be acquired. If not, then requests of Professor Ananth Jillepalli should be heard and followed up on in the conclusion and aftermath of our game’s launch with respect to general system, software, and hardware evolution over time.

*Solution Approach*

# System Overview

The project’s main functionality surrounds the idea of creating a complete game where all dialogue spoken by NPCs is generated in dynamic time by a generative AI, such as ChatGPT. The functionalities of this project can be broken into four main sections: Data Access and Conversion Layer (Persistence Layer), World Layer, the Game Environment, and Top-Level Input and Output Systems. Our design methodology is heavily influenced by the separation of different systems and boundaries that the data the game environment is working with will be crossed. The effect of this influence will be more thoroughly discussed in the Architecture Design section. In regard to the design of our data structures, there are two core data structures that need to be planned. The first is the data method for encrypting and masking the open AI keys needed to access the generative AI. These should not be stored locally and require special consideration. The other data structure is needed for storing the other game files and information that are accessed by the Unity Engine for all the other game information. The core consideration for designing our UI is creating stylized designs for menus and conveying other information to the player that is not immediately obvious without affecting the western theme present in the game or occupying too much screen real-estate and blocking view of the world environment. The specifics on component interfaces, data structures, and UI layouts will be discussed in the following sections.

# Architecture Design

## Overview

The Wicked West team has decided to use a Layered Model for this project. With the concept of our project, the data needed for our system will need to move through several different layers and be integrated from several different sources before being used in the game world and presented to the player. We will be integrating data, both from the games standard data and the open ChatGPT prompts that are generating dialogue for the NPCs. This data then needs to be processed and fed to various systems and scripts running inside the Unity engine for the generation of running of various parts of the game world. Our team saw the Layered Model as the most logical way to organize this process. Below is a component diagram that illustrates the architecture used to create our game system. The diagram is to be used as a guide for our team in our development process, for us to reference and use as a resource when planning our development. The diagram is also a visual representation of the component architecture that is decomposed in the following section. Here is an overview of the components, as seen in the diagram below. Starting from the bottom, both the save files and the ChatGPT server instance will be data items outside of the system we are building. The game state handler will connect to the save files and the world loader, allowing the saving and loading of files. It will also contain other game state information, such as active items and NPCs and their positions in the world. The ChatGPT server handler is connected to the ChatGPT server instances and the prompt state handler. It will handle the labeling of prompts, ensuring that they are active at correct times, and also connect the prompt state handler to the correct server instance for the character. The prompt state handler will connect to the server handler, the server instances, and the dialogue system. It is responsible for handling overall character prompts for giving to the generative AI, as well as passing player dialogue to the generative AI from the dialogue system and the AI’s in-character responses back to the dialogue system. The world loader connects to the game state handler and the world environment. It is responsible for loading the various parts of the world into the correct places, as well as all the underlying logic and functions that make the game world work as intended when the player interacts with it. The dialogue system connects to the prompt handler as well as the world environment. It is responsible for converting between player input dialogue and the responses generated by the AI and passing and loading that information to the world environment. The Game/World environment connects to the input handler and UI handler. It is responsible for maintaining the active game world that the player sees and interacts with. The input handler is responsible for collecting player input and passing it into the game environment, it only connects to the world environment and is the top-level input system. The UI handler only connects to the world environment, it shows parts of the world environment that are not immediately visible and prompts the player based on underlying logic in the game world. It is the top-level output system. Additional in-depth descriptions of these subsystems will be elaborated on in the following section.

A diagram of a computer

Description automatically generated

## Subsystem Decomposition

## [UI Handler]

### Description

The UI handler will focus on user interaction, specifically systems such as menus and player interactions. Menu icons such as character portraits and player crosshair will be updated in this subsystem.

### Concepts and Algorithms Generated

The UI Handler will use many sub-classes to work. For example, buttons on a pause menu would likely be their own section using scripts to execute events. Furthermore, UI elements may change when players interact in the game.

### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
|  |  |  |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
| updateUI | Game Environment |

## [Game Environment]

### Description

The game environment includes the world the player can interact with. This includes buildings, props, and NPCs. It will be the most important in the control of what the player can see and aesthetics.

### Concepts and Algorithms Generated

Being the most central part of presentation, the Game Environment subsystem will comprise of many classes that handle the looks of the game. It will receive inputs from the Input Handler depending on what the player interacts with.

### 2.3. Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| updateUI | UI Handler | Update the UI depending on player interaction. |
| dialogueReceived | Dialogue System | Receives player text from the game environment and updates the dialogue system. |
| worldInteract | WorldLoader | Logs changes to the world to be updated in the save file. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
| inputRecieved | Input Handler |
| updateDialogue | Dialogue System |
| updateWorld | World Loader |

## [Input Handler]

### Description

The input handler will handle controls of the game. If a player moves, the input handler will detect and send such to the game environment, updating the current point of view. It will also read interactions with NPCs and items.

### Concepts and Algorithms Generated

The input handler should be minimal, creating a class that will communicate with the game environment to ensure that inputs are properly read.

### Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| inputRecieved | Game Environment | Takes input received and sends it to the game environment to update interactions. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
|  |  |

## [World Loader]

### Description

The world loader will make the environment properly populated with props and NPCs depending on what is received by the game state handler. It will update on whether props and NPCs should be missing.

### Concepts and Algorithms Generated

The world loader is its own class that will update and store positions depending on what is used by the player, as well as updating during saving and loading so that the player can return with the world the same as it was.

### Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| updateWorld | Game Environment | Updates the world based on input received from the game environment as well as the game state handler. |
| changeGameState | Game State Handler | Logs changes and sends them to game state handler to update the world. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
| worldInteract | Game Environment |
| updateGameState | Game State Handler |

## [Dialogue System]

### Description

The dialogue system will allow players to interact with the NPCs using their own questions. Once the game environment receives a text from the player, it will generate a response using the prompts generated from the prompt state handler and update the text in-game to reflect such.

### Concepts and Algorithms Generated

As this is the controller for the dialogue system, the system will get the text from the player and send it to the prompt state handler, who will then update the prompt and send a generated text back to the dialogue system.

### Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| updateDialogue | Game Environment | Updates the dialogue sent to the environment as received by the generated language model. |
| sendDialogue | Prompt State Handler | Sends the inputted text to the prompt state handler. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
| recievePrompt | Prompt State Handler |
| dialogueRecieved | Game Environment |

## [ChatGPT Server Handler]

### Description

The ChatGPT server handler should handle all connections with the ChatGPT server. If there is ever a disconnect, the server handler should inform the game environment as such.

### Concepts and Algorithms Generated

The handler should be its own class and make sure there is always a connection with the ChatGPT server the game is active.

### 6.3. Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| serverUpdate | Prompt State Handler | Ensures the connection between a server and player. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
|  |  |

## [Prompt State Handler]

### Description

The prompt state handler will update NPCs prompts when either a question is asked from the player or the in-game story progresses. The prompt will then be sent to the dialogue system in order for an answer to be generated.

### Concepts and Algorithms Generated

Many subclasses will likely be used to control the eight NPCs we have planned for the game. For example, if one NPC were to die, then the system must be updated to remove any mention of that NPC from the other’s prompts.

### 7.3. Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| recievePrompt | Dialogue System | Provides an updated version of the NPC prompt to the dialogue system to use, as well as any errors. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
| sendDialogue | Dialogue System |
| serverUpdate | Server Handler |

## [Game State Handler]

### Description

The game state handler will handle saving and loading of the game environment, as well as communicating such information with the world loader to make sure that items and NPCs are in their proper positions.

### Concepts and Algorithms Generated

We will need classes for saving and loading, likely to a Json script for ease of use. This script will save in-game progress, such as story progress, which in turn will control NPC position and status, as well as the placement of items in the world and what could already be in the player’s inventory.

### 8.3. Interface Description

|  |  |  |
| --- | --- | --- |
| Service Name | Service Provided To | Description |
| updateGameState | World Loader | Update the world loader data after combing input from world loader as well as during the loading of a save file. |

Services Required:

|  |  |
| --- | --- |
| Service Name | Service Provided From |
| changeGameState | World Loader |

# Data design

There are going to be two main databases created with our application. One is a cloud server database run on Ubuntu 22.04 (LTS) x64. The need for this database is to encrypt and mask the OpenAI API keys used through an encrypted connection to our server, avoiding the dangerous practice of storing the API keys in the local files, which would then make them hash-able and obtainable to savvy users with malicious intent. This will hold user data when desired and be responsible for user authentication through NodeJS server-side code and a MongoDB database. Since this part uses JavaScript, the custom web API data will be encrypted (hashed) in a .env file on the host machine being run through our cloud hosting service hosting NodeJS. This data structure would come in the form of a primitive data type, and can be held in just a string, since that is the accessible location of our API keys in the .env file. The other non-primitive data types in JavaScript such as the lists and arrays (which will certainly be used) also in the package and in the dependencies of the package will be stored as such data structures in this database as well.

The other database is created and kept within the Unity local files stored on the user’s computer. As for the internal data structures within the Unity project build, following proper garbage collection practices will allow for data structures to be stored on a computer’s local files. Garbage collection in unity is automatic, though requires significant amount of CPU time. Our team will overcome this hurdle by storing our data wisely and optimizing our data structure usage for both performance and storage advantages. Memory leaks and programming errors will be handled through C#, our main language of choice for this project, due to C# having an automatic memory management system to reduce these risks [8]. Nearly all of the data stored locally will be stored in C# files (.cs) as files having non-primitive linear arrays, queues, and stacks. The files will also hold primitive data structures of floats, characters, integers, strings, and possibly doubles, however these primitive data structures will not significantly affect the garbage collection system due to the light storage sizes of the structures.

# User Interface Design

While we are still in the process of creating a 3D environment for our Wicked West, our team has created wireframes for the game menus that the players will interact with. These original wireframes are simplified to communicate the content in these menus, but the menus will be stylized in the final prototype to match the western theme of our game. Since our final product will be a video game, our team put lots of consideration into the items we include on our menus. We want our users to spend their time playing our game rather than clicking through our menus. Therefore, we’ve limited the items on our menus to include only what we deemed necessary for the user’s satisfaction and progress in the game.

Our game consists of two menus, a start menu, and an in-game menu. The start menu simply prompts the user to select if they would like to start a new game or resume their current game. The user may also have an option to update their game preferences on this page. The start menu is necessary for the implementation of the ‘Start Game’ and ‘Load Saved Game’ use cases. Once a user is in the game, they are able to pause the game and access the game menu by pressing the esc key. The game menu consists of six components in the form of clickable buttons (*Figure 1*): resume, journal, map, options, exit to menu, and exit to desktop. 'Resume’ returns the player to the game environment. ‘Journal’ allows the user to record thoughts, interactions, and observations from the game as well as view the game controls. ‘Map’ displays a map of the game environment to the user (*Figure 2*). ‘Options’ opens a submenu where a user can update their preferences and control the graphic and audio settings. ‘Exit to Menu’ exits the game and redirects the user back to the start menu. ‘Exit to Desktop’ will exit the game and close the application. The user will use the game menu to execute ‘Pause Game’ and ‘Access Game Instructions’ use cases.

When a user is playing the game, they have the option to interact with objects or NPCs within their area. To allow this, we will implement crosshairs (*Figure 3*) to inform the user how they are able to interact with that object, and the user can interact with the object by using a hot key such as ‘E’. Depending on the object, the user will be able to interact with the object, hide, or attack or break the object. If the user is within close distance of an NPC, they will see a ‘Dialogue’ option. Selecting the dialogue option will start a conversation with the NPC and direct the user to a conversation window. This screen will have three main elements, an image of the NPC the user is talking to, a text entry box, and an AI response. The user will use the text entry box to type and send their messages to the NPC. The AI response then acts as the NPC and sends a response back to the user. The AI response will provide clues to the user on how to beat the game. The crosshairs are used to implement ‘Start Conversation with NPC’, ‘Open Doors’ and ‘Pick Up Objects’ use cases.

*Project Testing and Acceptance Plan*

# Testing Introduction

## Project Overview

Wicked West is a game in which players will uncover the mystery behind disappearances in a fantasy western town. To do so, they will have to solve puzzles, hide from the monster, and most importantly talk and interact with the locals, who will respond to the player’s questions in real time using AI-generated prompts.

For testing, we must consider the game’s development in the Unity Engine. As such, we can rely on automated tests for features such as saving and loading. Certain story triggers can be tested too, including features such as the day-night cycle. However, most of our testing will be done in-game, testing user interface interactions with the world.

## Test Objectives and Schedule

The main objective of these tests is to ensure a player can perform a full playthrough of our game multiple times without failure. To do so, we intend to use the Unity Test Framework, allowing for both testing of scripts inside and outside of the game. For unit tests, we can use the framework to test all functionalities that are required to run the game. For game mechanics, we will likely need to test in Play Mode, in which we can interact with the in-game environment to test features. An important feature we must test is the ChatGPT integration, specifically having it respond to player input.

For the testing schedule, we will likely test as the game is developed. We can write unit tests for features that will not require player interaction to trigger, after which we can produce a deliverable of these tests. These deliverables will likely come at every major milestone in development, with more thorough reports delivered at the start. The second deliverable we will produce regularly will contain playtesting reports of the current build. This is for all the features that require user interaction to test, as not all features can be tested through code.

However, it is important to note that any testing involving ChatGPT responses will not be able to cover every situation, as AI can lead to unpredictable responses. As a result, we likely will not have gameplay related triggers tied to this dialogue, and as such any sort of testing with generative responses will be minimal.

# Testing Strategy

Testing will largely revolve around the gameplay itself. We need to be able to test key features of the game to ensure the player never experiences game-breaking bugs. Before this, we will unit test any features that don’t require direct player interaction through the Unity Test Framework, allowing for testing of C# scripts without always needing the game running. After which, we will debug through constant play testing and try to test most situations a player might come across. While it may be impossible to account for all situations, we will make sure to document. For our testing process, we will use continuous integration, as it would be efficient in our team in which we will largely work independently from each other.

**Developer Testing Process**

1. **Code Written:** Features and scripts will all be written in the Unity editor. This will be divided into specific features, such as player movement and interactions, as well as features that can be unit tested such as save load mechanics.
2. **Create Test Cases:** After the features are programmed, we will create test cases for each function that can be unit tested. For those that cannot, we can still create tests for playing specific situations.
3. **Run Tests:** After the tests are created, the tests should be run to find any issues.
4. **Debug:** If any issues are discovered through the test cases, they should be identified and fixed.
5. **Push Code to Remote:** The code is sent to the remote with the results of the tests.
6. **Create Pull to Main:** Once all tests have been passed, the update is pushed to main.
7. **Merge Branch:** The branch is merged, and the feature is added.

**Playtesting Process**

1. **Create Playtest Build:** A build for playtesting is created. Added features and their conditions for activation should be listed to know what to test.
2. **Playtest:** As we will likely be testing the game ourselves, we will playtest the build and look for any bugs. However, we may invite others to playtest our build, in which case we will take feedback from them.
3. **Receive Feedback:** We will record any issues and feedback from others.
4. **Make Changes:** We will update the newest build for playtesting.

# Test Plans

## Unit Testing

Our team will overall follow standard unit testing procedures; however, the unit testing methods will diverge in some areas specifically surrounding the Unity Engine. For these areas, we will be using the Unity Test Framework. This framework will allow for running of Editor Tests, which run code independently of the game engine for testing purposes, as well as Play Mode Tests, which run tests as coroutines to the actual game and allow for testing of specific features that involve interaction with objects outside of the script. For the code to be considered sufficiently tested, the team will be required to test all necessary game functionalities. These functionalities are all objects and scripts that would render the game unplayable if absent and is non-trivial. The team will also evaluate the relevance and impact of non-necessary units and decide if more tests must be developed. The units that will be evaluated are frequently used units that are not core game features. Individual developer discretion will be used when considering the extent of additional unit tests.

## Integration Testing

Due to our group working with the Unity Test Framework, our integration testing will mirror the methods described in the unit testing section with some limitations imposed by the system we are using. Due to constrictions created by working in the Unity Engine we will need to deviate from standard processes and test semi-isolated collections of objects and scripts that cannot always be completely integrated outside of full system testing. The unique structure and intended end result of the project may also make it difficult to integrate certain sections and scripts outside of full system testing. In cases where we need to deviate from standard systems, we will refer to unit testing results and developer discretion in the viability and time cost of integrating the specific sections.

## System Testing

System testing is a type of black box testing that tests all the components together, seen as a single system to identify faults with respect to the scenarios from the overall requirements specifications. The entire system is tested as per the requirements.

During system testing, several activities are performed:

## Functional testing:

Our team’s plan for functionally testing our game is to manually test our use cases. Our team of developers will implement and perform manual tests on our game. For the manual tests, our team will predominantly focus on testing the use cases outlined in the Use Cases section of the Project Requirements and Specifications document, which lists the developer and stakeholder expectations for our game’s functionality. Each functional requirement, ‘Start New Game’, ‘Load Saved Game’, ‘Pause Game’, ‘View Game Manual’, ‘Start Conversation with NPC’, ‘Exit Conversation with NPC’, ‘Pick Up Objects’, ‘Move Around’, ‘Hide’, and ‘Open Doors’ will have one functional test associated with it. Since we are using Unity to build out the game, we will be able to perform the functional tests by running a build of the game and performing the tests within the game environment. Each test will have a set of instructions that will be followed to perform the test, and our team of developers will manually validate the tests. In the event that a functional test fails, the developer testing the functional requirement should provide a description of the testing conditions to the development team and the test should reevaluate the test and correct the error.

## Performance testing:

Unity, the game engine we are using to develop our game, comes with a built-in tool for measuring game performance. The Unity Profiler tool gives in-detail game performance metrics including frame rate, memory usage, rendering statistics, and more. This will allow us to identify areas of poor performance which will help us provide the best user experience.

Our team will also test the performance of the non-functional requirements listed in the Project Requirements and Specifications document. We will use manual testing to test the performance of these components. When testing the performance of the non-functional requirements, our team of developers will evaluate whether each non-functional requirement was met, and should note any concerns with the performance, usability, or graphical issues. These concerns will be shared with the rest of the development team and will be reevaluated and corrected.

## User Acceptance Testing:

To test user acceptance, our team is considering recruiting a handful of potential players to test play our game before a final version of the game is completed. Members of the test group could be our classmates, friends, and family, and potentially our client. Having users play our game before it is released allows us to gather feedback from the user and uncover any bugs that were not found during functional testing. Our testing strategy for user acceptance testing is outlined below with sections detailing the required resources for the test players, the timeline for the testing process, and how our team will implement the feedback from the test players.

A. Required Resources

* A build of the game (to be distributed to each test player)
* A form for feedback (for each test player)

B. Testing Timeline

* A group of 2-5 test players will be chosen for the testing iteration.
* Each player in the testing group will be provided with the current build of the Wicked West game.
* Each test player will be given a full week to test the game.
* At the end of the testing time frame, our team will provide the test player with a form to gather the player’s feedback.

C. Implementing Feedback

* The team will gather and review the feedback submitted by all the test players.
* The team will meet to discuss any modifications to implement in the game based on the testing group’s feedback.

Our team will continue to reiterate through the process listed above until we receive positive responses from all the test players and there has been notable improvements to the game from the first prototype to the final build and all requirements have been tested and met. In order to get the most effective feedback, we should have different test players for each testing iteration as well.

# Environment Requirements

A computer capable of running the build of our game (PC, MAC, Linux) is required. For the computer, the performance of that computer needs to be excellent when available, as our game should not only be tested in common run states, but also on a computer that can indeed run our game with no dropped frames. In testing, the hardware and specifications must be isolated from the game, or else developers would not be able to determine whether certain errors and glitches came from the game, or from the platform/system it is being run on. The best and most optimal way to navigate this is to use a high-performance computer to run our tests on. That way, any lag or dropped frames can be inferred to come from the project’s performance, rather than the system. For tests run when high-end hardware is unavailable, the Unity Profiler tool will be used to help differentiate whether dropped framerate or memory storage issues come from the hardware or our project’s compiled build. The computer must be able to run Unity tools at a high performance.

Beyond this, most of the testing can be done without an internet connection. The black-box tests won’t need any, and the white/clear-box tests we will run do not require a connection, since our code is compiled locally through Unity. However, an internet connection will need to be maintained for testing the Ubuntu 22.04 (LTS) x64 server concealing our OpenAI API keys.

In terms of software, since our project is made in Unity, we will be taking advantage of NUnit. The easiest way of going about this is using the Unity Test Framework to create tests inside of assemblies which reference NUnit. Inside of our Unity project, we will create such assemblies to be “Test Assemblies”. These Test Assemblies are to be set up with the Test Runner UI in the Unity Test Framework (UTF). Tests are to be categorized into separate “Play Mode Tests” and “Edit Mode Tests”. Play Mode Tests can be run inside the Unity Editor or as a standalone. Edit Mode Tests are run only in the Unity Editor and are the tests which have access to both game code and Editor code. This requires a Tests folder to be created with its respective .asmdef file with it by default and will need the Unity development software to have references to the **nunit.framework.dll, UnityEngine.TestRunner,** and **UnityEditor.TestRunner** assemblies. The UnityEditor.TestRunner will be used for Edit Mode Tests. The other assemblies will be used for both modes of testing. Since Unity does not include NUnit, UTF, and user script assemblies when using the regular build pipeline, these will be included from the Test Runner window. TestAssembliesTests will be created as C# .cs files, and the NUnit Test attribute instead of the UnityTest attribute will be used for Edit Mode tests, aside from in cases where frame skipping or time-waiting will be tested via Play Mode tests. In other situations where we must yield special instructions, the IEditModeTestYieldInstruction interface can be used in Edit Mode Tests to implement our own special instructions, which will be needed in testing cases where custom yield instructions must be defined [13].

*Future Work*

For the future work to be done on this project, there are a variety of things that need to be done before the game is ready to be released. This section will go over the items that the team is aware of that need to be done.

### Inventory System Expansion

We plan for further expansion of the inventory system. This includes the ability to use items from the inventory on the environment for puzzle solving and information gathering, as well as checks for if certain items are in the inventory to enable events or changes in dialogue.

### Saving and Loading Updates

We will need continual updating to our save and load systems as more things are added. The possibility of switching saving plugins is now present in our plans, and that is discussed in more detail in the Alpha Prototype Demonstration section below.

### Event Tracking

We plan to track certain events for use in changing dialogue prompts given to ChatGPT and making changes to the environment. This tracking includes items picked up and deaths of NPCs.

### World Updates

We have seven more buildings that will be added to the world, as well as potentially some more items. There will also be changes to the game world floor, the layout of buildings, and the skybox.

### NPCs

There is intended to be eight NPCs in the game with their dialogue written dynamically by ChatGPT. These NPCs and the dialogue features still have much more work to go on with their implementation.

### Hiding Action

We intend to allow players to hide in certain objects to escape detection from the monster. The action for hiding and the items that can be hidden in still must be implemented.

### The Monster

Our monster is currently a stationary model. It still must be textured, animated, and given actions and pathing.

### Journal

In the service of providing information to the player, especially between gameplay sessions, we plan to implement a journal that allows for notetaking in game, or automatically does some of it for you.

*Alpha Prototype Description*

In testing of our alpha prototype, we have found that some components are working properly behind the scenes, and at least one component has been found to be working improperly behind the scenes which could affect us going forward if not addressed in the future.

Playthrough testing has shown that we have all functional requirements described in this document working in at least a prototype form. This was discovered by taking advantage of NUnit, and creating custom Commands where necessary to test our functional requirements being able to be referenced, accessed, and changed in-game during playthrough with our in-game developer testing console.

Note: For some more examples of in-game testing scripts created, see Figures 4-7 in the Appendix.

Further in-game testing with NUnit, the Unity Test Framework, and custom in-game commands has revealed that the Lazy Instance of our Game Manager is not being referenced properly with GameManager.Instance from within the game after Start during runtime. This will have to be addressed in the future so it can be properly referenced so no null pointer exceptions are thrown behind the scene in runtime.

Taking advantage of Unity, we used the built-in Memory Profiler Module, as well as the additional Memory Profiler package, to test our storage, memory, and framerate issues. This helped us isolate when framework issues were because of the system (example: a computer running our game is also livestreaming it), and when framework issues were because of incorrect gametime settings (example: Time.GameTime not being set to 1 when it should be).

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Testing with the Memory Profiler helped us reach helpful conlusions, such as our C# scripts not being memory-issues for our game. This can be told from looking at the Unity Objects in the Memory Profiler and seeing their % Total Memory taken up. The C# script MonoBehaviour is is most memory-requiring script, which is good and proper since it is a built-in Unity script for all Unity gameobejcts that want to interact with the Inspector and other Unity editor-related components. Since we do not observe our scripts above this script, and rather find our scripts at the bottom of this Memory Table, we know that we have used good memory practices in script-writing so far.

A screenshot of a computer

Description automatically generated

Note: Our custom AudioManager script is taking up less than the MonoScript component, which is reassuring especially since AudioManager deals with storing large audio files.

**Currently implemented subsystems:**

## UI Handler

### Functions and Interfaces Implemented

Our current UI has:

* Crosshairs for different interactions.
* Start menu.
* In-game schedule tracker.
* A player inventory.
* Pause menu with save, load, and options features.
* Conversation layout for NPCs.

Additions to be worked on include updated designs for UI buttons, a player journal, and in-game map.

### Preliminary Tests

A screenshot of a video game

Description automatically generated A white cylinder on a white surface

Description automatically generated

## Game Environment

### Functions and Interfaces Implemented

Our game world has:

* Multiple buildings with animations
* Enterable areas
* Working mine system
* Interactive items
* NPCs in the game world.
* Skybox.

Additions to be worked on include multiple skyboxes, updated scene transition areas, more buildings, ground textures, and updated building positioning.

### Preliminary Tests

A building with a red sky and clouds

Description automatically generated with medium confidence

## Input Handler

### Functions and Interfaces Implemented

Our current Input handler has:

* Working movement for keyboard.
* Jump button.
* Interactable key (talking to NPCs, picking up items)
* Inventory key.
* Pause key.

Additions to be worked on include making buttons more responsive, buttons for attacking.

## World Loader and Game State Handler

### Functions and Interfaces Implemented

Our current world loader can save the current game world. Load is work in progress as it can load through dev commands but not in-game yet. Plans include adding that functionality.

### Preliminary Tests

A screen shot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated



Note: pFieldMoveSpeed is our saved key, where p represents Player, and its corresponding value is the expected int of 10, which is the Player’s correct speed field value.

## Dialogue System, Prompt State Handler, ChatGPT Server Handler

### Functions and Interfaces Implemented

Our current dialogue system has:

* Generatable prompts to talk to NPCs with.
* NPCs responding to conversation.
* Connection with ChatGPT server.

Additions to be worked on include custom prompts the user can input.

### Preliminary Tests

A screenshot of a computer screen

Description automatically generated

*Alpha Prototype Demonstration*

During our prototype demonstration, we showcased all of the progress our team has made on our project so far. Our demonstration started from the Start Menu, where we were able to show some artwork and the functionality of the menu. After clicking the start button, we were redirected to our Load Saved Game Menu where players can choose whether to start a new game, or load a saved game, and are then placed into the game environment.

Once in the game environment, we started by demonstrating simple player movements. While demonstrating these player movements, we were also able to showcase the exteriors and interiors of all of the buildings we have modeled so far including the saloon, church, house, shed, and outhouse. The outhouse has door animations as well, so we were able to show how the door opens when a player approaches it, and closes when the player leaves the area.

We also gave a demonstration of the dialog system we have started to implement within the game. We highlighted how the crosshairs change when a player gets close to an NPC, and how we are able to generate barks for each character, which are a list of prompts that the player will be able to choose from to start a conversation with an NPC.

The next thing we demonstrated was the inventory system. First, we picked up all of the items we have included in the game so far by approaching them, and pressing ‘e’ when the crosshairs change. Then we viewed these items in the inventory by pressing ‘I’. Then we clicked through all the items in the inventory and their descriptions.

Finally, we demonstrated our Pause Menu functionality. We showed how to access the pause menu, and demonstrated the functionality of all the buttons included in the menu such as “Resume”, “Save Game”, “Main Menu” and “Quit”

One question/suggestion that we received is if the light source emanating from the player when tied to the mine was planned to be tied to an item. When mention of a flashlight-like object was brought up, it was suggested that we lean more period appropriate, with something like a torch or a lantern. Another suggestion that will be made in future implementation is to have the outside of the mine more visible when near the exit of the mine, so buildings visible from inside the mine could be seen in the scene. This was something planned to be done at a later date. Another suggestion we received was the suggestion of adding a run or sprint function to the player character for quicker movement. It was also suggested that we may want to add a stamina meter with that feature. It was also suggested that we add multiple skyboxes so that there were more fitting designs for day, night, and possibly sunset/twilight. The last suggestion we received was to create a more themed and unique ground for the world floor rather than the default Unity one. We had already planned to replace this as well.

Our mentor has been closely monitoring our project as it has been developed, so the majority of the questions we received were closer to suggestions. The items listed below here were items that were definite questions rather than suggestions for future work on our project. The first question we received was if the inventory was capped at a maximum of eight items. Currently the code is set to hold eight items, but our plan was to have the inventory capped at whatever number of items were available to be collected. At this point that number did not exceed eight, so there were only eight inventory slots. Another question we received was if the current building spacing and layout in the prototype was the intended final layout or only temporary. Our demo layout was temporary and created for easy showcasing and testing. Another question we received was how light and shadows were calculated. In this version, light either comes from the skybox or from interior lights in the building models.

Beyond the changes mentioned above that for the most part were already in our plans in some capacity, the largest adjustment to our plan that was discovered is the potential incompatibility between Easy Save 3, the plugin used for our save and load features, and Git version control. This incompatibility makes it difficult for certain parts of the game to be worked on in unison by multiple people and sabotages the file merging capabilities of Git. We will be seeking a workaround for this problem in the near future, and if one is not found, we will be replacing Easy Save 3 with some other saving and loading features that will be more compatible with the systems we’re using.

# Glossary

Application Programming Interface (API) – Software with a distinct function, specifically referring to the communication between a server such as OpenAI’s server and the interface the service is being provided on such as a Unity OpenAI API plugin like BitSplash’s OpenAI asset.

Barks – Phrases, words, and sounds from NPCs during encounters and triggered events. These items are usually filler text, such as things yelled at player characters when they walk by an NPC on the street.

DigitalOcean – A renowned cloud infrastructure provider known for simplifying the deployment and management of applications, websites, and services, offering robust scalability and developer-friendly features.

Domain Name System (DNS) – A decentralized naming system that translates human-readable domain names into IP addresses, facilitating internet communication by allowing users to access websites and services via a domain name rather than using numeric IP addresses.

Droplet – A cloud-based virtual server instance on DigitalOcean’s platform, designed for effortless deployment, scaling, and management of applications and services, offering flexibility and robust performance.

Generative AI - Artificial intelligence that creates data, such as text, images, audio, and 3D objects. It creates data by learning patterns from existing data of similar types then generating new outputs.

MongoDB – A document database with scalability and flexibility used for querying. [4]

NodeJS – An open source, cross-platform runtime environment for executing JavaScript code. [10]

Non-Player Character (NPC) – Any enemy or ally characters seen in a game that are not controlled by a player.

NUnit – A unit-testing framework for all .Net languages. Initially ported from [JUnit](https://www.junit.org/), the current production release, version 3, has been completely rewritten with many new features and support for a wide range of .NET platforms. [12]

OpenAI – An American artificial intelligence research laboratory which provides AI services such as ChatGPT API.

Premium (Intel) – Intel DigitalOcean CPU droplets which run second generation Intel Xeon Scalable processors featuring Cascade Lake architecture which operate at a base frequency of 2.50 GHz and max turbo of 3.90 GHz. [2]

Proxy – An intermediary server that acts as a bridge between clients and target servers, enhancing security, privacy, and performance through forwarding requests and responses which mask the client’s identity and location.

SSL Certificate – A digital certificate that encrypts data transmitted between a web server and a client’s browser, ensuring secure and private communication over the internet, enhancing trust and safeguarding sensitive information such as passwords and financial data during online transactions.

Ubuntu 22.04 (LTS) x64 – The long-term support (LTS) 64-bit version release of the popular Linux operating system which is renowned for its stability, security, and compatibility, suitable for a wide range of computing needs and hardware architectures. This version was released April 21, 2022, and is in long-term support until April 2027.

# Appendices

A screenshot of a computer screen

Description automatically generated

Figure 1

A map of a town

Description automatically generated

Figure 2

A black circle with white text

Description automatically generated

Figure 3

A screenshot of a computer program

Description automatically generated

Figure 4

A screenshot of a computer

Description automatically generated

Figure 5

A screenshot of a computer

Description automatically generated

Figure 6

A screenshot of a computer screen

Description automatically generated

Figure 7

# References

‌[1] IBM, “What is an Application Programming Interface (API) | IBM,” *www.ibm.com*, 2023. <https://www.ibm.com/topics/api> (accessed Sep. 19, 2023).

[2] “Introducing Premium Droplets with faster Intel and AMD CPUs and NVMe SSD,”

[*www.digitalocean.com*](http://www.digitalocean.com)*.* <https://www.digitalocean.com/blog/premium-droplets-intel-cascade-lake-amd-epyc-rome> (accessed Sep. 24, 2023)

[3] “Inworld Origins,” Inworld: The most advanced Character Engine for AI NPCs, <https://inworld.ai/origins> (accessed Sep. 17, 2023).

[4] MongoDB, “What Is MongoDB?,” *MongoDB*, 2019.

<https://www.mongodb.com/what-is-mongodb> (accessed Oct. 5, 2023).

[5] OpenAI, “About,” *openai.com*, 2023. <https://openai.com/about> (accessed Sep. 19, 2023).

[6] R. Barth, “The Convergence of AI and Creativity: Introducing Ghostwriter,” Ubisoft® - Official Ubisoft News, Previews and Features, <https://news.ubisoft.com/en-us/article/7Cm07zbBGy4Xml6WgYi25d/the-convergence-of-ai-and-creativity-introducing-ghostwriter> (accessed Sep. 17, 2023).

[7] “Square Enix AI Tech Preview: The Portopia Serial Murder Case,” SQUARE ENIX AI Tech Preview: THE PORTOPIA SERIAL MURDER CASE, <https://www.jp.square-enix.com/ai-tech-preview/portopia/en/> (accessed Sep. 17, 2023).

[8] U. Technologies, “Unity - Manual: Garbage collection best practices,” *docs.unity3d.com*.

<https://docs.unity3d.com/Manual/performance-garbage-collection-best-practices.html>

[9] Unity “Most Common Keyboard/Mouse Inputs for PC Games?” Unity Forum,

[forum.unity.com/threads/most-common-keyboard-mouse-inputs-for-pc-games.380594/](http://forum.unity.com/threads/most-common-keyboard-mouse-inputs-for-pc-games.380594/) (accessed Sep. 23, 2023).

[10] “What is Node.js? - Definition from WhatIs.com,” *WhatIs.com*.

<https://www.techtarget.com/whatis/definition/Nodejs> (accessed Oct. 1, 2023).

[11] Unity Technologies “Profiling Tools” *unity.com*. <https://unity.com/features/profiling> (accessed Oct. 21, 2023).

[12] “NUnit.org,” *nunit.org*. <https://nunit.org/>

[13] “Custom yield instructions | Test Framework | 1.1.33,” *docs.unity3d.com*.

<https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/reference->custom-yield-instructions.html (accessed Oct. 24, 2023).