**AI Assisted Game Development**

**Final Report**

*Developing a 3D Game While Incorporating AI*



**Mentor:**

Ananth Jillepalli, Washington State University

**Wicked West**

Megan Carver, Kyle Nepo, Nathanael Ostheller, John Kendall

**TABLE OF CONTENTS**

[I. Project Introduction 3](#_Toc162779916)

[II. Team Members & Bios 4](#_Toc162779917)

[III. Project Requirements Specification 6](#_Toc162779918)

[III.1. Project Stakeholders 6](#_Toc162779919)

[III.2. Use Cases 6](#_Toc162779920)

[III.3. Functional Requirements 11](#_Toc162779921)

[III.4. Non-Functional Requirements 13](#_Toc162779922)

[IV. Software Design 15](#_Toc162779923)

[IV.1. Architecture Design 15](#_Toc162779924)

[IV.1.1. Overview 15](#_Toc162779925)

[IV.1.2. Subsystem Decomposition 16](#_Toc162779926)

[IV.2. Data Design 20](#_Toc162779927)

[IV.3 User Interface Design 21](#_Toc162779928)

[V. Test Case Specifications and Results 23](#_Toc162779929)

[V.1. Testing Overview 23](#_Toc162779930)

[V.2. Environment Requirements 25](#_Toc162779931)

[V.3. Test Results 25](#_Toc162779932)

[VI. Projects and Tools Used 32](#_Toc162779933)

[VII. Description of Final Prototype 33](#_Toc162779934)

[VIII. Product Delivery Status 35](#_Toc162779935)

[IX. Conclusions and Future Work 36](#_Toc162779936)

[IX.1. Limitations and Recommendations 36](#_Toc162779937)

[IX.2. Future Work 36](#_Toc162779938)

[X. Acknowledgements 37](#_Toc162779939)

[XI. Glossary 38](#_Toc162779940)

[XII. References 40](#_Toc162779941)

[XIII. Appendix A – Team Information 41](#_Toc162779942)

[XIV. Appendix B – Example Strategy Reporting 42](#_Toc162779943)

[XV. Appendix C – Project Management 43](#_Toc162779944)

# 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.

Our primary client is Microsoft. However, at the start of our point of contact at Microsoft was no longer available, so Professor Ananth Jillepalli will serve as our client for the duration of the project.

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 the game, with the options available being enjoyable enough to add replayability. 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 Members & Bios

**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 Specification

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

## III.1. Project Stakeholders

When we were assigned this project, we were also assigned Microsoft as our client. However, when we began working on the project, our contact at Microsoft was no longer available to support us, and we were unable to secure a new contact. As a result, our team decided to continue on with the project as described, and Professor Ananth Jillepalli served as our client.

To explore the extent to which AI can be used for game development, we created a 3D video game that uses Chat GPT to play eight different non-playable characters. The main requirement of the project was to incorporate AI into the game, and we have met that requirement by implementing dynamic conversations between the player and the non-playable characters.

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

In summary, incorporating AI into our video game will aid in research regarding AI and game development. The player of our game will benefit from challenging levels and visually appealing graphics. Our team has treated these needs as we’ve built our game. The needs of our primary client, Ananth Jillepalli has been prioritized first, but the needs of the players have been considered throughout the design and development process.

## III.2. 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 ‘J’ key to pause the game and pull up the Journal |
| **Alternative Path** |  |
| **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 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 ‘Goodbye’ button to exit the conversation |
| **Alternative Path** | 1. Press the ‘Esc’ key to exit the conversation instead |
| **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 key to pick the object up |
| **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 key 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** | 1. The user presses the ‘Esc’ key to leave the inventory window. |
| **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 sprint |
| **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 |

## III.3. Functional Requirements

1. **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. |

1. **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. |

1. **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. |

## III.4. 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.

# Software Design

## IV.1. Architecture Design

### IV.1.1. 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

### IV.1.2. Subsystem Decomposition

1. **[UI Handler]**
   1. **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.

* 1. **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.

* 1. **Interface Description**

Services Provided:

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

Services Required:

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

1. **[Game Environment]**
   1. **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.

* 1. **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 |

1. **[Input Handler]**
   1. **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.

* 1. **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.

* 1. **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 |
|  |  |

1. **[World Loader]**
   1. **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.

* 1. **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.

* 1. **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 |

1. **[Dialogue System]**
   1. **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.

* 1. **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.

* 1. **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 |

1. **[ChatGPT Server Handler]**
   1. **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.

* 1. **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 |
|  |  |

1. **[Prompt State Handler]**
   1. **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.

* 1. **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 |

1. **[Game State Handler]**
   1. **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.

* 1. **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 |

## IV.2. 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.

## IV.3 User Interface Design

While in the process of creating a 3D environment for Wicked West, our team created wireframes for the game menus that the players will interact with. These 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.

# Test Case Specifications and Results

## V.1. Testing Overview

Our overall testing approach is to ensure that new features in branches can be built and tested in parallel to their merging to main, and to ensure this process of integrating tested features happens frequently. Testing during each process of merging features back to main helps us build confidence in the tested features of our games; Confidence in the existing system and features makes isolating failures in newly-implemented features easier. We are going to be using a form of Continuous Integration throughout the process of making our game, through making testing with each merge to main a practiced standard. Because our game is being developed in Unity, we will go about testing builds prior to accepting merge requests with the use of the QSFW.QC’s namespace’s [Command] feature. This, combined with the in-game developer console which runs in the Unity Editor and runs in project builds, is our primary way of writing in-game tests for our project in C# code\*. Because the dev console is an in-game feature added to aid in testing, it helps primarily for writing Unit Tests and Integration Tests. Throughout the process of development, bugs and failures that have been found will be reported to the group by being brought up verbally at a meeting, in the discord, or through our repository’s Issues feature. An additional process of patching prior issues found throughout the process will occur in the days leading up to a monthly Sprint report, and additional patches can be made then. However, if we follow the process of Continuous Integration described above, bugs and failures found post-merge that are not integration-related should be minimal in existence. The Unity Test Framework (UTF) will be used to run performance and some system-level testing. This will allow us to run system-level tests referencing NUnit according to how they are written to be tested, if desired.

Many of our Unit Tests have been able to run through the editor’s log, whether in the Unity Editor or in a build of our project. These tests will most notably consist of Debug error-logging code. Unit Testing has been implemented well and in a simple manner so far, and error-logging can be expected to continue as the primary method of Unit Testing throughout future development of our project. If the Unit Test needs a bit more complexity, the [Command] feature can be combined with the editor’s log and used to trigger in-game Unit Tests at specific times desired.

Our Integration Testing process has also been going well, as the connections between different units, modules, and instances within our scenes can be tested using the [Command] feature combined with the in-game developers console. This allows for use to run tests in-game, while:

1. Writing [Command]s to trigger relationship events in-game to occur and testing exception cases.
2. Utilizing the Unity Editor’s ability to add and remove components and objects from the scene in-game, allowing for references between our modules to be Integration Tested.

This has been going well, and the above methods will be our primary method of continued Integration Testing throughout development.

System Tests are to be written in accordance with the team’s and Client’s expectations of the project. System Tests will be verbally discussed in team meetings, in order to get clarity on expected specifications of the desired test, and then written in-game. The method of doing so depends on the complexity of the desired System Test. For example, in fixing an Issue relating to our New/Load Game Selection screen not displaying its proper saved details, an Issue was created and it was discussed in a meeting with our team that Name, Day, and Time would need to be tested to be fixed. This was thereafter implemented within our SaveLoadScript.cs file with the help of written [Command]s for testing such as SaveName(string idDesired), CheckLoaded(string keyToLoad), and DelKey(string keyToDel), in order to get to both a log message displaying the correct day, time, and name, but also visual confirmation in-game. This method allows System Tests to be written according to desired procedure, and then implemented in Editor Mode or a project build.

Functional Tests can be expected to be tricky, as they will deal with the connection of the user to either:

1. Our provided, server-protected API key.
2. The user’s own API key.

Part b) has been suggested recently, and we are working on a certain future implementation of the method at the end of the project. Part a) has been planned for late development since the beginning of the project. Testing these two functionalities will be two of our greatest feats of Functional Testing, as testing that the users can access the system securely is important. This late implementation will need a team meeting to discuss the specifics of implementation prior to writing these important, needed Functional Tests. Other more minor forms of functional tests have been written in our project throughout prior testing processes.

Performance Tests in our project will utilize the Unity Profiler tool for analyzing performance. Issues with memory allocation/usage have arose from adding too many objects in the scene, and the Profiler has been able to help us isolate the objects causing performance issues. In terms of testing rendering, we will have to move on to a more advanced form of exploring graphics rendering issues using Unity’s Frame Debugger. This can be anticipated because we can be expected to need implementation of a rendering system soon following recent object overpopulation issues. The Frame Debugger will then be used to show a visualization of the rendering performance once implementing the rending system, giving developers more insight in the build of the rendering system. Performance benchmarks and the Unity Test Framework has allowed us to write [UnityTest]s using the System.Diagnostics namespace’s features like stopwatch, allowing for the creation of code-triggered Performance Tests in-game.

User Acceptance Testing has been done to a small scale and degree so far, and is done through the expectations of our Client. Prior meetings have yielded expectations from our Client about specific attributes desired on features, leading us to have team meetings to discuss what should be expected. This has been a form of User Acceptance Testing through planning System Tests; However, it can be expected that as our project reaches late-stage development, more specific desires may be brought up by our Client, since more features are to be integrated throughout development. For this case, we will be sending a demo in the future to Ananth Jillepalli and using one of our 12:00pm meeting times to discuss specifics he desires to see following playthrough, and write tests in our team meeting on the following Thursday.

\*Note: In terms of testing Unity project builds specifically, the actual build file does not need to be pushed to the Git repo, as it is expected to maintain above 5GB throughout development, and each user with a Unity build themselves can create a build of the project on their local devices with code pushed to main. Thus, while it is expected for builds to be tested prior to being pushed to main, the build files themselves will not be pushed to main- only the new testing code written and possible other Unity file changes.

## V.2. 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].

## V.3. Test Results

UI Handler Tests:

*Example: Testing of days being loaded as expected on startup from a Save File on Day 1.*

A screenshot of a computer

Description automatically generated

Discovered:

* Previous failures in inventory and pause menu integration (Has since been patched).
* Issues with inventory or canvas objects being in the GameManager Instance through Inspection-Assigning references (Patched).
* Start Menu integration issues with displaying proper day, time, or Name/ID (Patched).

Game Environment Tests:

*Example: In-Game developer console test to set it to night mode in skybox testing.*

A screen shot of a computer program

Description automatically generated

*Example: Testing of when a new destination point is set for the monster*

A screen shot of a computer

Description automatically generated

Discovered:

* Buildings need to be resized to fit player, while still avoiding doors (Patched).
* Player should run faster (Added).
* Buildings should be more maneuverable (Added).
* Items can be picked up, and the game can save a specific list of items picked up in its logic.
* NPCs should receive models (Added).
* NPCs should receive idle animations (Being implemented).
* Skyboxes can indeed change in-game, and during a fade.
* Monster should always loosely path towards player/objective (Added)
* Monster’s should be more difficult to escape from (Being implemented)

Input Handler Tests:

*Example: Log Error testing during NPC interaction, also logged in the developer console.*

A screenshot of a computer

Description automatically generated

Discovered:

* Movement with added sprint is working, and faster now without issue.
* Jump is working properly.
* Pause is working properly.
* Inventory and Pause keys logic was not integrated together or setup well (Patched).
* Interact key properly interacts with objects and NPCs.

World Loader and Game State Handler Tests:

*Example: Testing Spawning in at the expected, loaded location (when Load File exists)*

A screenshot of a computer

Description automatically generated

Discovered:

* The saving system for saving GameObjects as references with EasySave3 is not git-friendly.
* The workaround for this was troubleshooted, and its been found that saving key-val pairs with EasySave3 is git-friendly.
* Thus, while operational, we’ve decided at a team meeting that this will be something implemented once all attributes (keys) to be saved are known, to save us time from manually adding saves to removed/scrapped features throughout our process.
* T\_Teleport to sky, the Teleport script itself, and loading in on saved locations have been removed.
* Locational load-in and all other load/save features will come in a future update, in the form of key-val saving attributes on scripts needing so.

A screenshot of a computer

Description automatically generated

Discovered:

* Work prior was as stated: “Load is work in progress as it can load through dev commands but not in-game yet. Plans include adding that functionality.”
* Now, worlds can be loaded from New Game button on an outside scene properly for a fresh game state.
* Worlds can be loaded from a Saved State, provided once exists, which currently includes an update of location, days passed, time, name, player fields, and a copy of the player’s picked up listed inventory.
* Game State handler was needed, so it exists under the Game Manager now.

Dialogue System Tests:

*Example: Implemented User Discussion with the Salesman*

A video game screen capture

Description automatically generated

Discovered:

* The dialogue can break if you try to purposely currently, and reveal to be a ChatGPT bot.
* The NPCs make up details occasionally, so we will need to plan to patch that with use of temperature adjustments, and test those adjustments.
* Our current system is not fail-safe to unexpected results/failures from the OpenAI system/integration as a whole.
* We will need to be careful in how we test the dynamic Dialogue System, including keeping track of proper token usage, language model testing, and more. Team meetings will be used to discuss further findings.

User Acceptance Tests:

At this point we have not had a playable demo that is functional enough to run without one of the developers there to supervise and troubleshoot. At this point we have now created a playable demo and are passing it out to several testers for playtesting. We will have results from that soon.

# Projects and Tools Used

Include a summary of the libraries, frameworks, and tools you used to implement your project.  
For example, you could have used some web frameworks, a database, a network message  
passing tool, graphics generation libraries, and various operating system platforms. Please list  
these out with a short one sentence note about what it was used to build/support in your project.

|  |  |
| --- | --- |
| Tool/Library/Framework | Quick note on what it is for |
| PixelCrushers Dialogue System for Unity | This was used for dialog handling, as well as AI API integration. |
| Easy Save 3 | Saving and loading features |

As a quick survey, let me know what languages you wrote some of the project in. This is  
anything you wrote yourself, not just used in libraries. This includes both programming  
languages and markup languages.

|  |  |  |  |
| --- | --- | --- | --- |
| Languages Used in Project | | | |
| C# | Language 2 | Language 3 | Language 4 |

# Description of Final Prototype

Our final prototype will demonstrate the capability of AI in games with 8 NPCs that are able to converse with the player. Furthermore, the game should include all features that were planned, such as having a day/night cycle, a working menu system with saves, and a monster that will hunt the player during the night. Finally, the game should be complete with both a lose and a win condition.

For specifics, the player will have full access to the in-game town. During the daytime, players can talk to NPCs and solve puzzles that will grant them access to items to assist the player. The player can enter buildings in which some NPCs will reside. The objective of the game is to find all key items and survive until the last day or all are collected. A journal and map menu is accessible to provide player’s information on their current story progress. Players can save at any point and return to the save if necessary. Players can only interact with a limited number of NPCs per day, after which the game will switch to a nighttime state. For the prototype, a debug menu will also be accessible in case the player runs into issues.

A screenshot of a video game

Description automatically generated

Photo of in-game UI, including debug menu.

The player can start conversations with the NPCs. Doing so will create a text dialogue box on the top for the NPC response as well as a dialogue prompt on the bottom for the player to type any sort of dialogue to interact with the NPC with. The purpose of this is to allow NPCs to give hints to the player on how to acquire story related items and the background of the story. Over in-game time and as the story progresses, the prompts given to the NPCs may change which may have them receive new information or even take away knowledge of other NPCs. For prototype purposes, an API key will be provided to allow for ease of testing, but the final build may have players use their own key.

A screenshot of a video game

Description automatically generated

Demonstration of dynamic conversations players can have with NPCs.

During the nighttime, the player will be able to save NPCs from the monster by finding the location of the deceased NPC and finding their lifeforce. Doing so will aggravate the monster, which will then start to chase the player down. The player can hide in designated places such as in barrels or outhouses. The night ends when the player returns to their shed. If the player is caught by the monster, they lose a life, and the game ends when the player loses all three.

A video game with a skull and a bottle

Description automatically generated with medium confidence

In-game items and monster (static model) on display. Items are interactable.

User Manual:

How to install:

* Download the zip file provided.
* Extract file.
* Run the build executable.

# Product Delivery Status

The project has been and will continue to be published for free open access and play at <https://wicked-west-project.itch.io/wicked-west> Our project was demonstrated to the general public at the capstone presentations as well as our mentor during a meeting with him.

The source code can be found at <https://github.com/WSUCptSCapstone-F23-S24/msft-aiassistedgamedev>.

The project can be accessed at the itch.io link, where it can be downloaded and ran as a standalone executable. From the source repository it can be loaded and ran in Unity in an editable version.

# Conclusions and Future Work

## IX.1. Limitations and Recommendations

One of the plainest limitations of the current prototype is some of the lack of general refinement and overall testing. With more time to develop and test, we would be able to add more features and build more on the features that have already been implemented, but due to the time constraints of the overall project there are going to be things that go unfinished or are not as finished as we would like them to be.

The major limitation of the proof of concept we have developed is that unless studios are developing in-house generative AI, the characters or other items that are being constructed or assisted by generative AI are limited to whatever restrictions are imposed by the creator of the AI system. In our current project we are using the GPT-4 ChatGPT system. Some of the creator-imposed limitations on that system are intentionally designed to limit sensitive content, which could pose problems for creative storytelling using that system. We have already discovered that our system will not respond whatsoever to profanity and restricts the responses generated by the character to also not contain any. This also extend to other sensitive content, such as anything beyond a vague mention of violence or death as well as any mildly suggestive or sexual content. While these content restrictions that we are seeing work for games with lower ratings, any games that wish to tell more graphic or dark stories would be severely limited by the restrictions of the AI systems. This would particularly impact genres such as Horror, Dark Fantasy, Cyberpunk, as well as many others. One other notable issue with these restrictions is that they are not guaranteed to be consistent. Items could be changed after production and release of a game making one of the topics unallowed or restricted due to events making the content sensitive or general perception changing, similar to how many topics from older media would be less accepted if it was made today.

Another issue comes from the dynamically generated text. Due to the nature of how dialogue is generated without oversight from the developers, things are shown directly to the player without any sort of proofreading by the developers. This can result in contradictions, confusing information, statement of items beyond the scope of the project, as well as general lack of understanding from the AI, with things such as stationary characters telling a player to follow them. These all detract from the final experience of a game and can impact the enjoyment that the player has and change their experience as well as the quality of the final product for the worse.

There is also a cost associated with generative AI. Due to the game needing constant contact with the AI’s servers to run it turns what should be a standalone singleplayer game into a product that requires internet connection and server usage, which adds cost on the end of the developer and an inconvenience to the player. These costs also grow very quickly with lots of players and could quickly cause an outage if a sudden spike in players passed the previously set cost threshold.

## IX.2. Future Work

Overall, our biggest conclusion from this project is that generative AI can be used for game development, but it is likely not super feasible for active use in the methods we’re using on a large scale. It is very effective at writing for characters, and most of the lines that we have seen so far would work very well with minor human correction. The most viable future work with this project would be continuation of testing, improvement, and expansion of the already existing game. There are some features that likely will not be fully complete or exactly match our vision at the end of the project, so future work would likely just be finishing or adding to the game. In terms of commercialization for our product, we have already published it on itch.io. This allows us to share the game and allow players to try it for free with an option to donate if they wish to. The donations would go back into the server costs for the product. The main platform for publishing PC games is Steam. Steam and many other similar platforms require a fee paid upfront to publish a title. Due to the smaller scale and experimental nature of our product, as well as the lack of funding, we have decided not to publish on Steam or any similar platforms at this time.

# Acknowledgements

Ananth Jillepalli – Thank you for mentoring and working with our team throughout the project, as well as acting as our stand-in client.

Lexi Young – Thank you for creating the original concept for the world Wicked West is set in and allowing our team to build on that world.

Our Testers – Thank you for playtesting our game and helping us gather data to improve it.

# 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.

# 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).

# Appendix A – Team Information

A group of people posing for a photo

Description automatically generated

*From left to right: Megan Carver, John Kendall, Nathanael Ostheller, Kyle Nepo*

**John Kendall**

Contact: john.kendall@wsu.edu

GitHub: ImJohnKendall

**Kyle Nepo**

Contact: kyle.nepo@wsu.edu

GitHub: KyleANepo

**Megan Carver**

Contact: megan.carver@wsu.edu

GitHub: megancarver06

**Nathanael Ostheller**

Contact: nathanael.ostheller@wsu.edu

GitHub: quickbick

# Appendix B – Example Strategy Reporting

To test our game, we had a handful of play testers who tested the list of functional requirements to ensure functionality as expected. The play testers also tested for new bugs by playing the game in ways it was not intended to be played. Our play testers raised many new bugs and gave helpful feedback about the gameplay by participating in conversation with one of our team members, or by filling out a google form which we linked on our game’s itch.io page.

# Appendix C – Project Management

Our team participated in 30 minute weekly meetings with the client, as well as hosted team meetings with just the project team. During the fall semester, we met with the client from 10:00am to 10:30am on Tuesdays, followed by a team meeting. During the spring semester, our team met with the client from 12:00pm to 12:30pm on Wednesdays, and hosted our team meetings in-person on Thursdays at 1:30pm.

During our client meetings, we would discuss the issues that each team member worked on during the last week, and made plans for what we were going to work on the next week. During our team meetings, we discussed design decisions, delegated tasks, troubleshooted bugs or merge conflicts, and asked each other questions that pertained to the individual work we were doing.

Our team used Discord as our main communication tool. We used discord to ask each other questions, resolve merge conflicts, and update each other on our progress throughout the week. During the fall semester, our team used discord to meet virtually for our team meetings as well.

All of our team member’s tasks and contributions were logged using GitHub. Every week after our client meetings, our team would create Git issues to assign to themselves for all the tasks they wanted to work on over the next week. These issues were also put onto a project board, and assigned low, medium, or high priority.