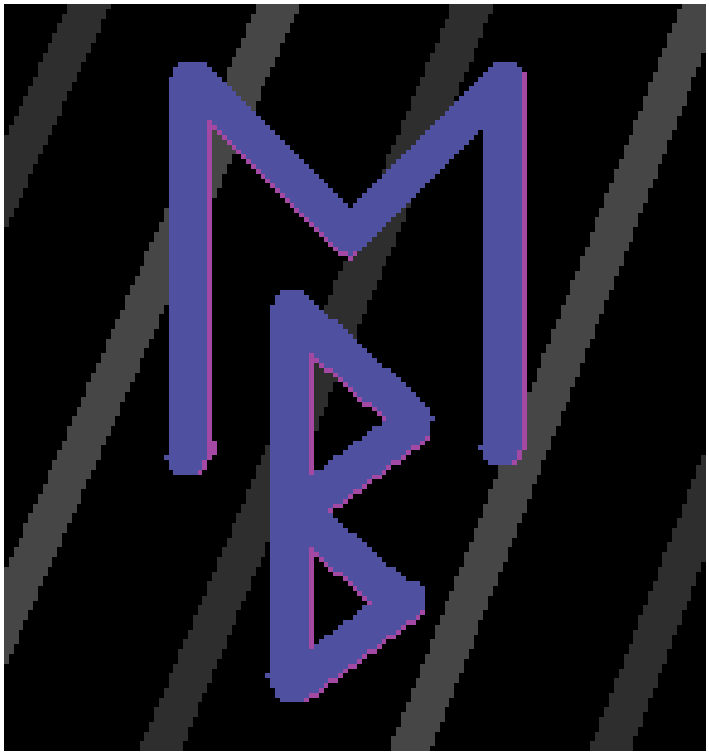
Hunter Black

Mechbot Documentation



Executive Summary

Mechbot is a 2D platformer video game for PC using the mouse and keyboard for controls. In Mechbot, the player controls a robot character to explore a facility in order to collect unique weapons and fight enemies. The player character and enemies have a health and damage system to allow for real-time combat. The player has an energy system that refills over time, while each enemy has a unique fighting pattern to threaten the player. Each weapon has a primary fire mode using the mouse left-click button, as well as a special fire mode using the mouse right-click button which will expend an amount of the player's energy to deal higher damage and provide a unique effect. Spikes will present a hazard to the player, instantly killing the player when touched.

The player starts with three lives to beat the level. There are two checkpoints that save the player's progress in the level when walked over, allowing the player to respawn at that checkpoint when defeated at the cost of a life. If the player loses all three lives before beating the level, they will Game Over and have to restart the level from the beginning. The player beats the level by defeating a boss character that has a special spawning and fighting pattern to present a higher threat to the player. Defeating the boss character takes the player through a brief victory sequence before returning them to the main menu. When the player plays again after beating the level, they will start from the beginning of the level with three lives with the weapons they had unlocked when they beat the level.

Main Menu Scenario | Provide player with a menu

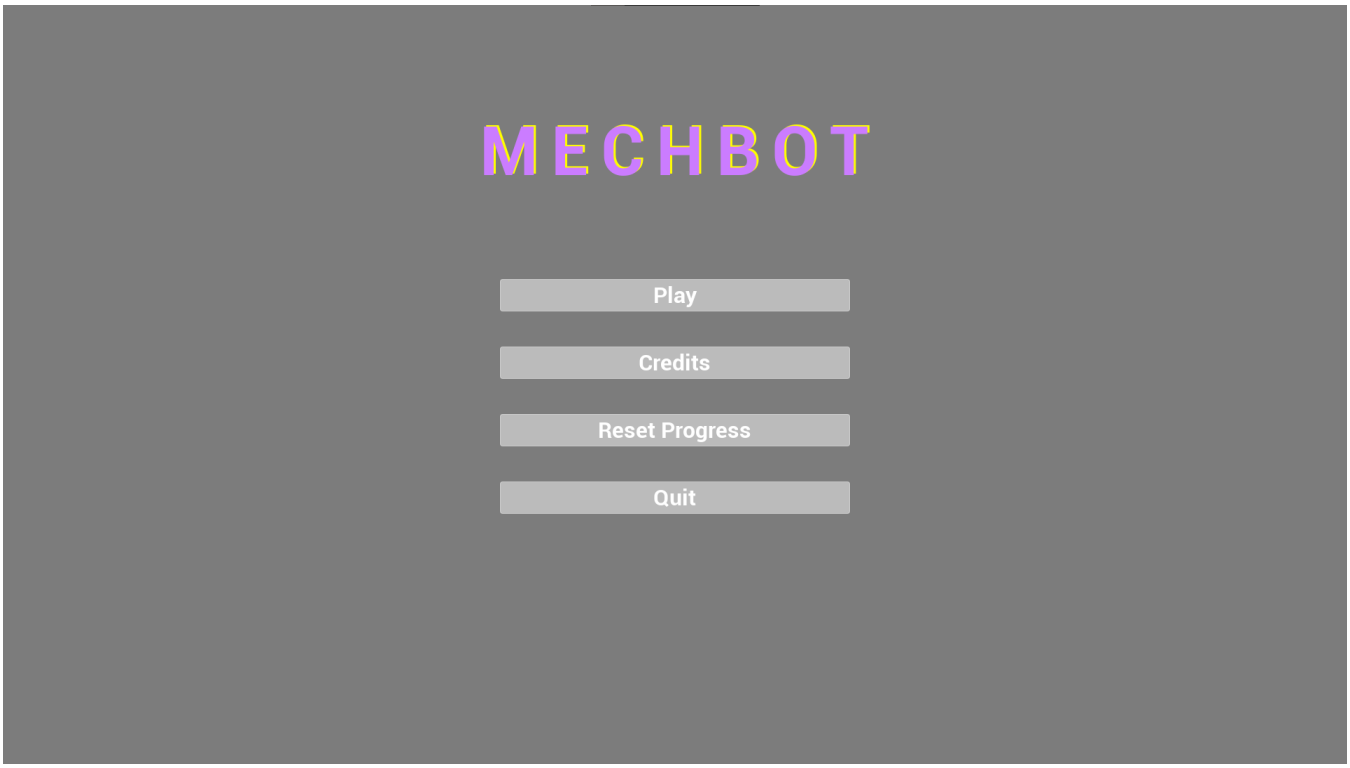
On the screen, the title of the game is displayed, along with “Play”, “Credits”, “Reset Progress” and “Quit” buttons, which will provide functionality when clicked by the player.

“Play” – Loads the player into the game’s level, starting the game.

“Credits” – Displays the authors of the assets used, along with a link to the asset page.

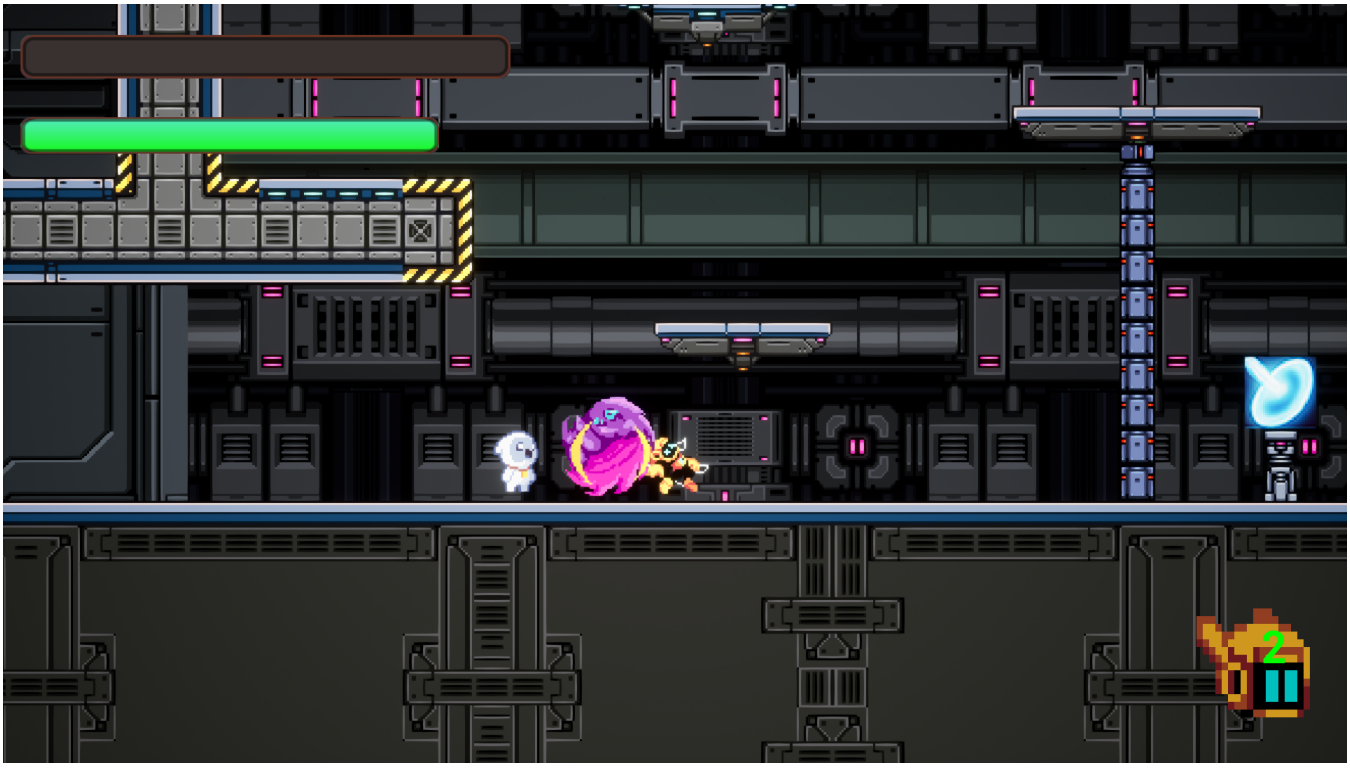
“Reset Progress” – Resets the player’s progress in the game.

“Quit” – Quits out of the game back to the desktop.



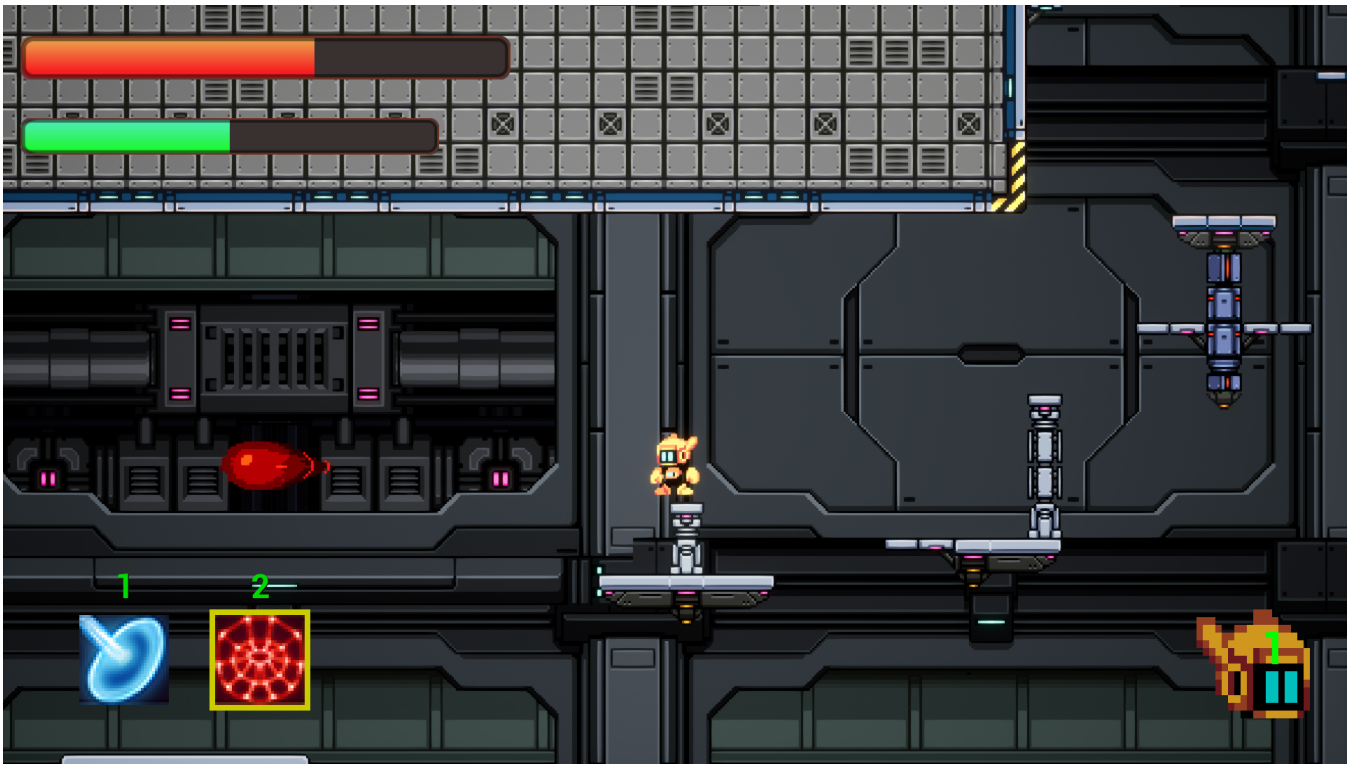
Combat Scenario | Enemies and Player fighting

The enemies use unique methods to attack the player. Upon landing an attack on the player, the player will take damage, decreasing the player's Health. The player can likewise attack enemies to decrease their health. An enemy or player that loses all of their Health enters a death sequence.



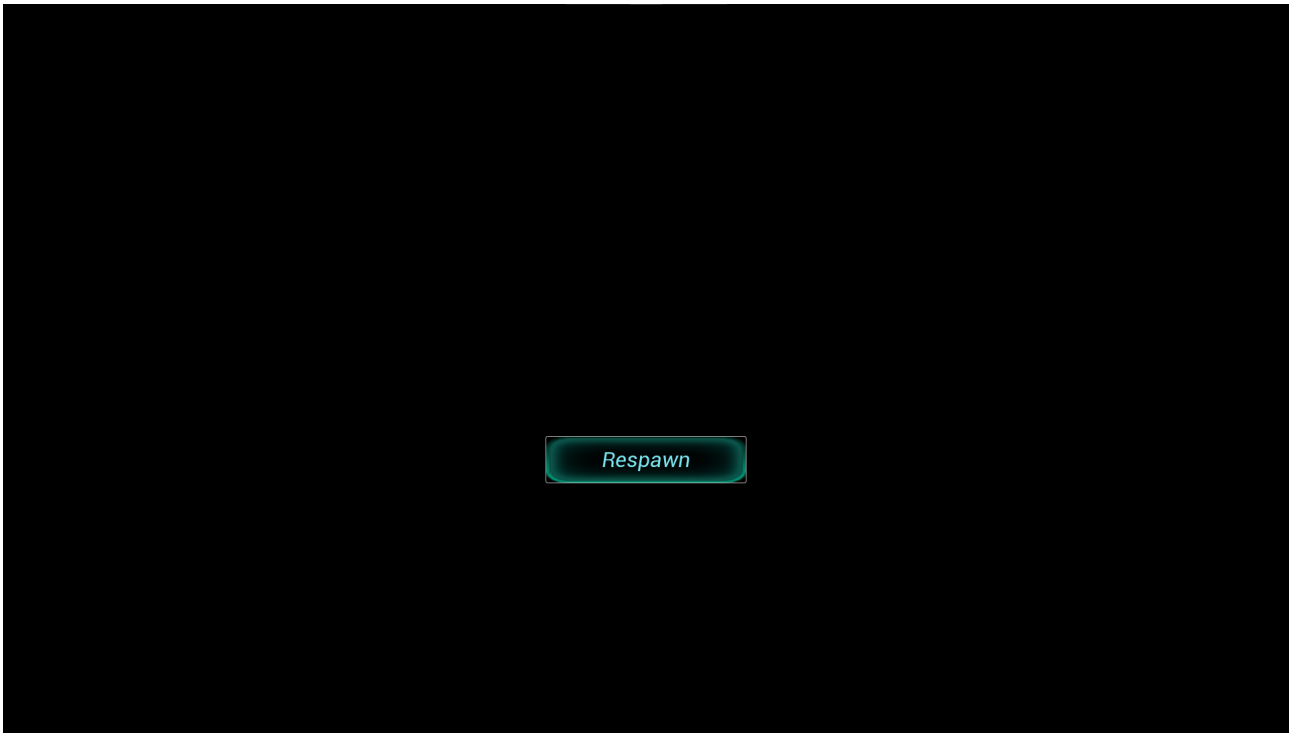
HUD Scenario | Display the player's information

The HUD shows the player's Health (red bar top-left), Energy (green bar top-left under Health), Weapons collected (bottom-left), and Lives (bottom-right). This information is updated live when any potential interaction occurs, such as the Health bar updating when the player takes damage. The numbers above each Weapon represents the key the user needs to press to equip it.



Death/Game Over Scenario | Displays when the player dies

When the player's Health reaches 0, the screen fades to black and displays a "Respawn" option if the player has at least 1 life remaining, or "Game Over" and a "Main Menu" option if the player has no lives remaining.



Reports

The game records debugging logs using Unreal Engine's UE_LOG feature to keep track of any errors the player may experience. When the game is run using the Unreal Engine editor, these logs are displayed in the Output Log window and saved to the project's directory, under the "Mechbot/Saved/Logs" folder. There are no additional reporting features for this project at this time, but future updates for this project may introduce some to allow players to report bugs or feedback to the developer.

System Architecture

The source code is kept in two main locations: the "Mechbot/Source" folder and the "Mechbot/Content" folder. "Mechbot/Source" contains the C++ files, while "Mechbot/Content" contains all assets used and Blueprint code.

Source Code Structure

Source code structure introduction. The following is a summary of the source code directories and their contents:

Directory	Usage
Binaries/Win64	Generated by Unreal Engine. Contains files to run the Editor and Shipping.
Config	Contains project settings files for the game.
Content	Contains assets for the game (such as sound, textures, UX, and levels) and Blueprint code.
Intermediate	Generated by Unreal Engine. Contains files to build the project in Unreal Engine's Editor.
Executable	Contains the executable (stored in a .zip file) for the game.
Saved	Contains files automatically saved by Unreal Engine. Could potentially be excluded.
Source	Contains all of the C++ source files.

Highlighted rows indicate directories containing source code.

Executables

Executables are stored in "Mechbot/Executable/Mechbot.zip" to adhere to GitHUB's file size limit.

Mechbot (Mechbot.exe)

This executable is the game itself. Running this will run the full project, giving players the ability to play the game through to completion.

UEPrereqSetup_x64 (UEPrereqSetup_x64.exe)

This executable was generated by Unreal Engine upon creating the "Mechbot.exe" file. There is not much information about it online, but it appears to be an executable for installing any missing requirements (such as certain .NET packages) to run the game.

Code Architecture

The architecture for this project is that logic related to assets is stored in Blueprint code in Unreal Engine's Editor, while other logic is stored and handled in .cpp files. The project's folders are categorized based on their object type to keep the directory organized.

Database or Data Store

Player data is stored in a SaveData game object. Outside of the functionality provided by Unreal Engine, it specifically stores the player's lives, collected weapons, and the checkpoint the player should spawn at. The game is paused while data is being read or written to prevent data corruption.

Unreal Engine saves the data by saving it to the memory buffer, then writing that buffer to the file. Conversely, it loads the data from a save file by saving the file's contents to the memory buffer, then passing that data into the code as data.

External Files & Data

Unreal Engine generates many files when creating a game. This is true for both the code compiled as well as when the executable game is generated (contained in “Mechbot/Executable/Mechbot.zip”). The files in the .zip file are mostly .dlls to make the game compatible with Windows and certain graphics cards. Similarly, the folders and files in the Mechbot directory not contained in the Source or Content folders are automatically generated by Unreal Engine to allow the compiler to build the Unreal Engine Editor.

Programming Language | C#.NET

Mechbot has been developed using Unreal Engine and C++. Unreal Engine provides some of its own C++ code in an “Engine” project, which any developer can use. The C++ code that has been developed for this project is in the “Mechbot” project, which is also found in the Source and Content folders. Some UX interaction logic and enemy AI logic has been developed using Unreal Engine’s Blueprints, which is a visual programming language focused on connecting nodes to execute C++ and Blueprint code.

Upon generating the Mechbot project, Unreal Engine created a series of folders and files along with the executable file. These generated files appear to be third party .dll files to ensure the game runs.

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

MechPaperDroid | MechPaperDroid.h/.cpp

This class stores all data for living game entities, such as health, whether the entity can attack, and functions for taking damage.

MechPaperPlayer | MechPaperPlayer.h/.cpp

Inherits from MechPaperDroid. This class stores all data specific to the Player, such as energy, weapons, and a unique death sequence.

MechUtility | MechUtility.h/.cpp

A generic class for storing objects with a main activation and special activation, giving a base functionality for both, and giving the Utility to the Player.

MechWeapon | MechWeapon.h/.cpp

Inherits from MechUtility. This class stores the main and special bullet types and spawns the relevant bullet type based on which activation method was used.

MechTool | MechTool.h/.cpp

Inherits from MechUtility. Not implemented, but kept for future updates.

MechGameInstance | MechGameInstance.h/.cpp

This class stores persistent data to be carried over every time the player respawns, such as the player's lives and active checkpoint.

MechSaveData | MechSaveData.h/.cpp

This class stores all data to be saved to and loaded from a file, including the player's lives, collected weapons, and current checkpoint, as well as the functions to commit the saving and loading.

MechCheckpoint | MechCheckpoint.h/.cpp

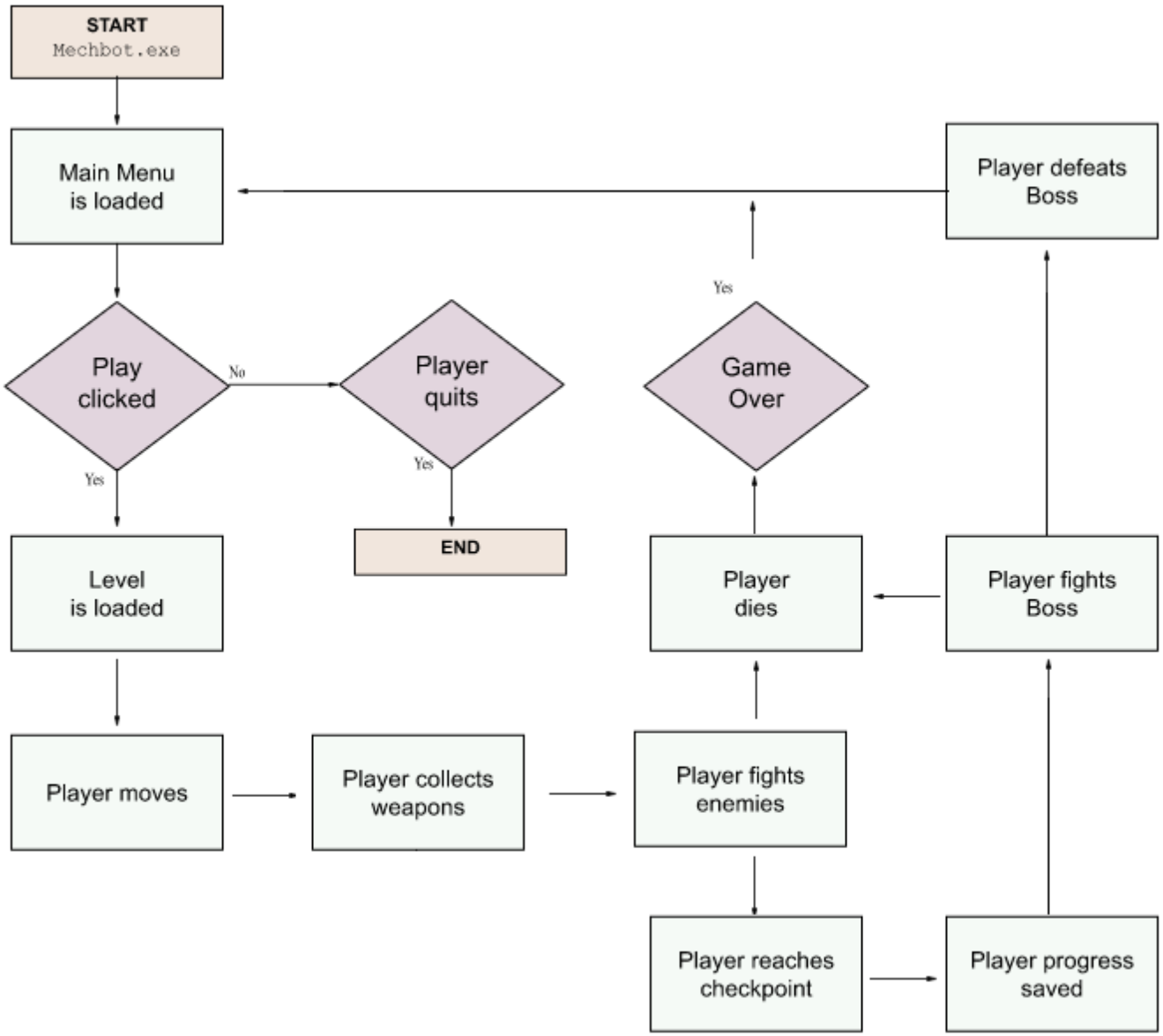
This class stores an Id to know where to spawn the player, as well as a function to save the player's current progress.

Project Modules

This project does not utilize any project modules. Unreal Engine may have some modules built in to its Editor, but these modules are not inherently a part of the Mechbot game itself.

Program Start and End Flow

The program's execution flow is mostly handled through Unreal Engine since there are many aspects that go into starting up a game that can be condensed into the same control across most games. From the user's perspective, the flow would look something like the following:



Summary

Mechbot is a 2D shooter platformer game written in Unreal Engine and C++ that utilizes an actor system for storing health and enabling combat capabilities. It saves player progress through different circumstances, such as when the player dies or when a checkpoint is reached. The game ends once the player loses all three lives or defeats the boss of the level. The C++ source files can be found in the "Mechbot/Source" folder, while the Blueprints and assets can be found in the "Mechbot/Content" folder. The structure for these folders are organized into folders based on the object type.

APPENDIX B (BUILD AND RELEASE PROCESS)

New builds and updates will be provided in the GitHub page along with a changelog file discussing the changes. It will require users to pull the project every time they wish to update, which will reset user progress. A future implementation of the project may utilize a more seamless method for updating.

APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

To run Mechbot, the user will need to download the “Mechbot.zip” file under “Mechbot/Executable/” and extract its contents. Once extracted, the user can run the “Mechbot.exe” file. It may prompt the user to install additional files. If the game does not run and it does not prompt the user to install additional files, the user will need to navigate to the extracted “Engine/Extras/Redist/en-us/UEPrereqSetup_x64.exe” file and run it to download any further prerequisites to run the game. The user should then be able to run the “Mechbot.exe” file.

APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

To work on the code, developers must download and install the latest version of Visual Studio 2022, then download the Mechbot project from the GitHub repository. Once these two steps have been completed, the developer needs to navigate to their Mechbot directory, right-click on the “Mechbot.uproject” file, then click “Generate Visual Studio project files”. This will open a command prompt window, which will close automatically once finished. Once the window automatically closes, the user can open the “Mechbot.sln” in Visual Studio.

Once the Mechbot solution has been opened in Visual Studio, the developer should open the Solution Explorer window, right-click “Solution ‘Mechbot’” and select “Build Solution”. This may take a while. Once complete, the developer should run the project in Developer mode to build the Engine Editor, which may also take a while the first time it is run. Once the Unreal Engine splash screen closes and the Editor is opened, the developer can work on any in-Editor files (such as assets or Blueprints). The developer can also make changes to the C++ code, but any changes to the C++ code will require the developer to close out of the Editor and re-run the project to see their C++ changes take effect.