Mystic Maze

NEA 2021-2022

Table of Contents

[1 Analysis 4](#_Toc95133131)

[1.1 Introduction 4](#_Toc95133132)

[1.1.1 Proposal 4](#_Toc95133133)

[1.1.2 Video Games 4](#_Toc95133134)

[1.1.3 Roguelikes 5](#_Toc95133135)

[1.1.4 Supervisor 6](#_Toc95133136)

[1.1.5 Critical Path 6](#_Toc95133137)

[1.2 Investigation 7](#_Toc95133138)

[1.2.1 Creating The Survey 7](#_Toc95133139)

[1.2.2 Survey 7](#_Toc95133140)

[1.2.3 Target Audience 12](#_Toc95133141)

[1.2.4 Related Games 13](#_Toc95133142)

[1.3 Objectives 17](#_Toc95133143)

[1.3.1 Main Objectives 17](#_Toc95133144)

[1.3.2 Specific Objectives 17](#_Toc95133145)

[1.3.3 Planned Technical Skills 20](#_Toc95133146)

[1.4 Restrictions 20](#_Toc95133147)

[1.4.1 User requirements 20](#_Toc95133148)

[1.4.2 Limitations 22](#_Toc95133149)

[2 Design 23](#_Toc95133150)

[2.1 Overview 23](#_Toc95133151)

[2.1.1 Languages / Libraries 23](#_Toc95133152)

[2.1.2 Objective Flowchart 24](#_Toc95133153)

[2.1.3 Human Computer Interaction Diagram 26](#_Toc95133154)

[2.2 Classes & Functions 27](#_Toc95133155)

[2.2.1 Main Classes 27](#_Toc95133156)

[2.2.2 Class Diagram 27](#_Toc95133157)

[2.2.3 Main Functions 27](#_Toc95133158)

[2.2.4 Function Relationships 27](#_Toc95133159)

[2.3 Game processes 27](#_Toc95133160)

[2.3.1 “Main” method 27](#_Toc95133161)

[2.3.2 Spritesheets 27](#_Toc95133162)

[2.3.3 Vector Normalisation 28](#_Toc95133163)

[2.3.4 Collision Detection 28](#_Toc95133164)

[2.3.5 Game States 29](#_Toc95133165)

[2.3.6 Saving/Loading Data from Files 29](#_Toc95133166)

[2.3.7 Cooldowns and Delays 29](#_Toc95133167)

[2.3.8 Animation 29](#_Toc95133168)

[2.4 UI Processes 30](#_Toc95133169)

[2.4.1 Button 30](#_Toc95133170)

[2.4.2 Drawing Text 30](#_Toc95133171)

[2.5 Character Processes 30](#_Toc95133172)

[2.5.1 MP Regen 30](#_Toc95133173)

[2.5.2 Player Animation 30](#_Toc95133174)

[2.6 Enemies Processes 31](#_Toc95133175)

[2.6.1 Enemy AI 31](#_Toc95133176)

[2.6.2 Enemy 1 31](#_Toc95133177)

[2.6.3 Enemy 2 31](#_Toc95133178)

[2.6.4 Enemy 3 31](#_Toc95133179)

[2.6.5 Boss AI 31](#_Toc95133180)

[2.7 Map processes 31](#_Toc95133181)

[2.7.1 Floor Generation 31](#_Toc95133182)

[2.7.2 Room Generation 31](#_Toc95133183)

[2.8 Items 31](#_Toc95133184)

[2.8.1 Active Items 31](#_Toc95133185)

[2.8.2 Passive Items 31](#_Toc95133186)

[3 Technical Solution 32](#_Toc95133187)

[3.1 A/B/C Cross Reference 32](#_Toc95133188)

[3.2 Main Game Programs 33](#_Toc95133189)

[3.2.1 Main.py 33](#_Toc95133190)

[3.2.2 Player.py 34](#_Toc95133191)

[3.2.3 Sprites.py 37](#_Toc95133192)

[3.2.4 Maps.py 38](#_Toc95133193)

[3.2.5 UI.py 39](#_Toc95133194)

[3.2.6 Item.py 40](#_Toc95133195)

[3.2.7 Enemy.py 42](#_Toc95133196)

[3.2.8 Framework.py 43](#_Toc95133197)

[3.2.9 Constants.py 44](#_Toc95133198)

[4 Testing 46](#_Toc95133199)

# Analysis

## Introduction

### Proposal

I am trying to create a potential product that will captivate and fully engage the user and provide a worthwhile experience that is worth the time and money, they would spend on it. Due to this general outline, I have endeavoured to look for an active form of entertainment that will garner the user’s full attention.

I have in conclusion come to the main premise for my NEA project being a top-down action, adventure, bullet hell roguelike video game product that will feature multi-floor randomly generated levels, randomly generated enemy/item spawns, fluid combat, and a large map to explore.

### Video Games



Figure

Video games are a form of entertainment that was created with active participation from the user in mind. Many video games feature similar attributes which are staples of the industry such as heads-up displays, stats, etc. Video games are extremely universal and can be used to deliver many different experiences from gripping stories to intense action and frantic gameplay. This has led it to become a mainstay of household and online entertainment.

Due to these factors, I have decided to make a video game as I am trying to make a product that can grab the user’s attention and sufficiently challenge them. It’s also a suitable form of content for what I am looking to achieve with this project as it is an active use of a player’s time and they have a choice of what happens during each run through the game, giving the player agency over the occurrences during their experience and in turn control over what they want to do for their fun.

### Roguelikes



Figure

Roguelikes are a genre of video game born from the original video game Rogue (Figure 2) which introduces the permanent death of the player once their health reaches 0 resulting in the game subsequently starting over again and randomly generated content which can be seen through the sprawling maze-like maps and unpredictable enemy and item spawns of the games from the genre.

I am making a roguelike as they are a great way to introduce replay-ability into a game due to the lack of progression that happens solely within the game, while most of it is apparent in the player’s knowledge and execution of manoeuvres in specific scenarios. The randomly generated content also adds an air of unexpectedness and allows for a broad level of fun that a player can have which is uniquely their own. Roguelikes are also known for their difficulty that comes hand in hand with the perma-death attribute of the genre as to stop players from steamrolling the game on the first try. Most roguelikes employ have knowledge and skill barriers which a player would build as they play the game.

### Supervisor

My supervisor is going to be Mr. Abbas, my computer science teacher and head of the department who will observe and guide me as I progress with my project throughout the stages. I will be sending my project sections as I complete them on Microsoft teams and upload programs, I do this for my technical solution through GitHub for him to assess as I work on it throughout the time we have been allotted for the technical solution.

### Critical Path

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Requirements** | **Time to do it** | **Date Done** |
| **Analysis** |  | - |  |
| **Design** |  |  |  |
| UI |  |  |  |
| Character |  |  |  |
| Enemies |  |  |  |
| Map |  |  |  |
| Items |  |  |  |
| RNG |  |  |  |
| **Technical Solution** |  | - |  |
| Testing UI processes |  |  |  |
| Testing Character  processes |  |  |  |
| Testing Enemies  processes |  |  |  |
| Testing Map  processes |  |  |  |
| Testing Items  processes |  |  |  |
| Testing RNG  processes |  |  |  |
| **Testing** |  | - |  |
| **Evaluation** |  | - |  |
| **Final Submission** |  | - |  |

## Investigation

### Creating The Survey

### Survey

To aid in the conceptualization of this project I have decided to ask potential users themselves about their opinions, preferences, and criticisms of the genre and video games in general. I have conducted a survey to collect concrete data to inform my design decisions and the game’s direction/objectives. This will influence my project as the information would help guide my vision and keep my ideas within the scope of an end-user who would potentially play the game.

<https://docs.google.com/forms/d/1STTIMnRdvlOQ8mdrwHO4SkaF44b480vjgLId_8_fpfQ/viewanalytics>

Chart, bar chart

Description automatically generated

Figure

This question helped me gauge what devices an end-user would prefer to play on and from this, I can infer that to develop this for the pc is to be the main priority with development for mobile being a possible path I could take to create cross-platform compatibility which could simultaneously increase the complexity of the project as I’d have to develop for 2 versions (IOS and Android) of a completely different system. It would also introduce the use of new libraries or software’s which could invertedly slow down production making it important to focus on one platform for now.

**Chart, pie chart

Description automatically generated**

Figure

Asking this allowed me to better assess the average quality of players’ computers to inform If I should massively improve my optimisation approach when I am developing the game. From this, I can assume that because 23.6% of my users have average or bad setups, I should make sure that I don’t make the game unnecessarily memory heavy by avoiding inefficient implementation of programs while having leeway for more system heavy algorithms if I need to due to 47.1% having great setups and 29.4% having amazing setups.

**Chart, timeline, bar chart

Description automatically generated**

Figure

From this I can tell that most video game players have a high preference towards action games and adventure games with votes at 81.3% and 75% respectively in favour of them with shooters being just behind at 68.8% in favour. With this in mind, I’ll focus on making sure combat is as smooth as I can make it with an expansive map to satisfy those who like the adventure of a video game. I’ll also ensure that magic combat feels satisfying and enjoyable to satisfy shooter game players.



Figure

Responses to this question reinforced the opinions from the previous one, where players listed games, like Hollow Knight, Valorant, Terraria and Team Fortress 2, adhering to those genres. I’ll endeavour to explore what makes those games succeed and see what I can implement into my own game to improve it overall.



Figure

This had particularly detailed answers referring to many different positive’s players looked for in games. Some mentioned how it was an escape from reality when talking about adventure games and others talked about time consuming, replayable games being a driving factor for their interests.

Text

Description automatically generatedText

Description automatically generated

Figure

To satisfy these players interests I’ll make sure to emphasise the roguelike aspects of the game and build upon basic mechanics I introduce to make them more engaging and satisfying. I might also consider a form of new game +, to increase the length of a run if a player is able to successfully beat the game once, which would scale up the difficulty and add new challenges.

Chart, bar chart

Description automatically generated

Figure

From these responses I decided what I’m going to relate my games mechanics on the most with Realm of the Mad God with 37.5% of people voting for it (although it may be a result of bias due to where I posted this survey) while Enter the Gungeon and Hades tied with 25% along with Risk of rain at 18.8%. I explain in more detail what I plan to take as inspiration for my game in the “Related Games” section.



Figure

This question provided me with a lot of insight into what users specifically prefer when they consider their favourite roguelikes. Some responses talked about achievements and “arbitrary goals” to reach which are staples in successful roguelikes as to be a sign of skill. I’ll develop a skin system that unlocks as you complete specific challenges set in game to provide a superficial progression system within the game.

Graphical user interface, application

Description automatically generated

Figure

Others mentioned build diversity and cool combinations. I’ll use these to create fun and creative differences between each run by adding a form of weapon manipulation and perhaps passive item synergies. If I have time, I could also introduce different classes with their own strengths and weakness to further build on the variance between runs.

Text

Description automatically generated with low confidence

Figure

The last things players mentioned was responsive and fluid combat. I will address this by creating algorithms to tackle weird movement glitches and allow for fully movement in all directions for both the player and the weapon as the game will be function as a twin-stick shooter. I’ll also try to implement vector approximation programs to improve this even further by working off the inputs of the player.



Text, letter

Description automatically generated

A screenshot of a computer

Description automatically generated

Figure

From these responses I can examine that more choice and more tense moments can help bolster my games moment to moment action. I can also make use of varying effects and debuffs to create that extra level of micromanagement for the player. In depth gameplay and mechanics are what keep a roguelike afloat due to the fact that there is no “end” in the majority of roguelikes.



Text

Description automatically generated



Figure

Many people found the main appeal of Zelda games to be the grand adventure and the use of your items along the way with nice and solid combat and movement. I’ll try to also provide moments of highs and lows to increase a player’s enjoyment. I may also consider the idea of changing the basic idea of the game to be a rogue lite in where permanent changes to the game are available but kept to a minimum.

### Target Audience

From my survey I can ascertain that my potential target audience own average-great pc setups, have a respectable level of skill when it comes to roguelikes specifically. I can also assume that an end user would have knowledge of games of the same breed and should be able to be acquainted easily with the format. However, to make sure I don’t leave outliers of the statistics locked out from playing the game I’ll endeavour to create a lightweight game with some tooltips to provide some guidance to new players and low spec pc users alike.

### Related Games

#### A Link to The Past

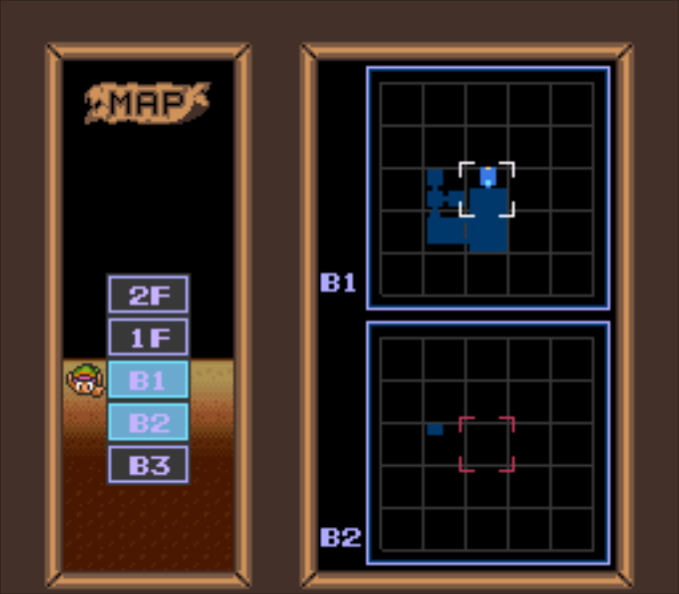
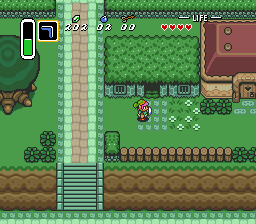


Figure 15

“A Link to The Past” is an action-adventure video game that, to many, embodies the spirit of a traditional adventure. It has consistent gear that grows with you along your journey and has great puzzling gameplay that pulls together the entire idea of untethered exploration. This along with the aesthetic and game design behind the dungeons helps create a great experience that has stood the test of time.

I plan to use the multiple floor concept that it uses in its dungeons to create a feeling of wonder and anticipation. I could also utilise the puzzle features along with this to help create a complete experience. To help build on the roguelike aspect of the game I will use random generation in the map’s formation. I will also try to introduce individual room generation and perhaps semi random puzzles.

#### Enter The Gungeon



Figure

“Enter The Gungeon” is a 2D roguelike that takes the tried-and-true formula and raises the bar with a large pool of weapons that completely define each run, each with their own special applications, hence creating a level of strategy to use specific weapons at the right time for the most effect which rewards players that learn from previous experiences with much more player freedom and expression in their subsequent runs. The weapons also have synergies that activate whenever you have specific passive/active items changing and improving them which introduces a whole new world of possibility and entertainment.

To improve my own game, I am going to use the diverse playstyles available in the game to my advantage while introducing some modular capabilities to them if I have the time. I will also use the ideas behind the different characters available to build on the potential idea of having different classes within my own game.

#### Risk Of Rain 2



Figure

Risk of rain 2 is a 3D roguelike that makes use of its surroundings and passive items that scale and complement one another within different tiers to create a simple system of what is good and what isn’t for new players to tell the difference between as soon as they start. Outside of this it uses quite a lot of the standard roguelike formula to create a simple but endlessly re-playable game.

From this game I plan to use the simple introduction to items and weapons through item tiers to make the game more new-player friendly and ease in new players, pushing down the bar to entry for most roguelike games and tapping into a new market of players.

#### Realm Of the Mad God

A picture containing map

Description automatically generated

Figure

Realm of the Mad God is an 2D MMORPG rogue-lite that uses basic RPG stats and addictive loot and shoot bullet hell combat to create a wholly unique experience unlike any other. It has an abundance of classes that differ in their gear, capabilities and effectiveness ensuring players will find something they will like to stick with. Each class has their own specific special ability that fundamentally defines them and is the drawing factor of the class. Dungeons have modifiers and items can be modified as well introducing another level of detail on top of the fun moment to moment gameplay found through its mechanically difficult bosses.

To build on my game I will use the bullet hell nature of the game to my advantage to create challenging combat that will make gameplay more tense and impactful. I will also introduce the ability to have modifiers to dungeons and even weapons through an optional mode unlocked by completing the game once to avoid the game being unnecessarily hard for new players.

## Objectives

### Main Objectives

1. The Game must have a user interface (UI) so that the player can have a unintrusive aid to improve their control and understanding of the game.
2. There must be a character that can be controlled by the user to act as their avatar in the Game so that they can interact and influence the world around them. This character must be able to fight, explore and move objects as well as other actions unlisted here.
3. There must be Enemies the Player can fight and inhabit the map with to provide challenge and an opposing force that will help improve the stakes during gameplay. There would be multiple different types of enemies
4. The Map must be randomly generated and consist of multiple floors to provide the User with ample space to explore and hone their skills while also maximising the number of occurrences RNG can affect the run.
5. The Items must be either Active (need to be used to grant an effect) or Passive (always grant an effect as long as the player has it equipped) as well as providing bonuses when certain items are used in conjunction with one another.

### Specific Objectives

These objectives outline the projects specific information and clauses to help detail my approach when it comes to the design stage. This may very well change as I progress and think of new ideas, or my original ideas change and morph to fit new problems that will come up.

1. UI
   1. When the player opens the game there will be a Main Menu that contains 3 buttons: a button to play, a button to enter a settings screen and a button to quit the application.
      1. The settings screen will have sound effects sliders and music sliders to decrease/increase volume.
      2. The settings screen will have keyboard settings that when clicked on can be changed by pressing a key.
      3. The settings screen will have “save & apply” button as well as a “return to main menu” / “return to pause screen” at the bottom of it.
   2. When the player presses the play button there will be a screen that has a character skin select, play game button and a difficulty slider.
      1. Skins would unlock after completing specific in game challenges.
      2. Difficulty slider would modify multiple aspects of the game including enemy hp, map size, quantity of difficult rooms etc.
      3. When the start button is pressed you would be brought into the game.
   3. When the player is in game, they would see a health bar, a mini map and a row of boxes containing active items.
      1. When the player’s character takes damage it’s health value would decrease and be displayed visually on the health bar.
      2. There would be a hotkey to hide the and unhide the mini map in game.
      3. When the player hovers over active/passive items with their mouse an info box appears detailing its specific details.
   4. There would be a hotkey to open the pause menu which would have a button to open the settings screen, return to the main menu and quit the game.
   5. The rest of the screen would be reserved for the main game.
   6. There would be a hotkey that when the player presses it the game screen would close, and it would transfer to the map screen.
2. Character
   1. When the game starts the character would be created at the root of the map.
   2. The character can move in all directions and shoot in all directions.
   3. The character would have stats for health, speed, defence, attack, and dexterity.
      1. These are modified by passive items.
      2. Attack effects the damage of the attacks, Dexterity affects the attack speed, Defence reduces the amount damage taken and speed increases movement velocity.
   4. The character would start off with a basic wand.
      1. This wand would shoot at an average speed with low damaging shots.
   5. The character loses health when they take damage.
   6. When the character’s health value hits 0, the screen changes from the game screen to the game over screen.
      1. The game over screen would have two buttons: a button to return to the main menu, a button to enter
   7. The character is animated based off a sprite sheet
   8. The sprite follows the direction that they are moving in (so when moving up the sprite is facing and moving up)
   9. If shooting the sprite follows the mouse direction and does something according to the active item depending on what item is equipped.
3. Enemies
   1. There will be 5 enemy types in the game (subject to change)
   2. The first would be an enemy that orbits the player and shoots.
   3. The second would be an enemy that follows the players co-ordinates with a delay and shoots wide arcing shots with a short lifetime to provide pressure to the player.
      1. It would also be able to follow the player through rooms to provide more of a challenge if the player decided to speed through the dungeon by keeping aware of their surroundings.
   4. The third would be a tank like enemy that cannot be attacked on one side and can adjust its undamageable side depending on the location of the player from time to time
   5. The fourth would be a long-range enemy that would stay a certain distance away from you in a circle and shoot from afar.
   6. The fifth would be a charging enemy that would dash to your position and place a mine before retreating. This would then explode after a second or two but the enemy has low health so it can be avoided with smart play.
   7. There would also be a boss that would utilize the enemies attack patterns as well as using some room wide area of effect attacks.
      1. It would also barrel towards the characters general direction
4. Map
   1. There are multiple floors to each map ranging from 1 to 3.
   2. To finish a map/level you must find a boss key that allows to get into the boss room and finish it
   3. There are 6 room types
   4. The first is a normal room which has a decent chance for items to spawn after killing a set amount of enemies
   5. The reward room is a room that has a free item with no strings attached however it can sometimes be locked and need a key which would mean the item is better
   6. There is a risk for reward room with a huge number of enemy spawns which could end a run but there are a lot of items to gain.
   7. There is the boss room which is the only way to end the run, outside of dying, and the key is found after clearing 75% of the dungeon where an enemy would begin to get a chance to drop it.
   8. There are also secret rooms that would require a consumable to open as they are usually hidden with a few visual clues
5. Items
   1. Items fall into either being active or passive.
   2. Active items are either weapons or consumables
      1. Weapons all have a recast time, a damage number, and a shot speed with other specifics of the weapon being in the weapon description
      2. Consumables can be scrolls, potions etc. with a set amount of uses
   3. Passive items are general stat changing items with some of them effecting main gameplay, e.g., a passive that makes all shots bounce from walls.
   4. Items drop from rooms after being cleared at a decent chance and from enemies at an extremely low chance.

### Planned Technical Skills

#### Model

* Simple data model in database (Dictionaries and Multi-dimensional arrays)
* Simple OOP model
* Hash tables, lists, stacks, queues, graphs, trees
* Single-dimensional arrays
* Appropriate choice of simple data types
* Complex user-defined use of object-orientated programming (OOP) model (Classes, Inheritance, Composition, Polymorphism, Interfaces)

#### Algorithm

* Graph/Tree Traversal
* List operations
* Advanced matrix operations
* Recursive algorithms
* Complex user-defined algorithms (optimisation, minimisation, scheduling, pattern matching)
* Dynamic generation of objects based on complex user-defined use of OOP model
* Simple user defined algorithms (a range of mathematical/statistical calculations)
* Generation of objects based on simple OOP model
* Simple mathematical calculations (average)

## Restrictions

### User requirements

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure

From the Steam statistics which collects data on the systems of players around the world, the average pc specifications are rather high with 4 core CPUs (36.79%) running at 2.7-2.99 Ghz (17.71%) and a GTX 1060 (10.14%) as well as 16GB ram (49.69%). Although these may be high according to my survey, I can conclude that I should try to make my game as low spec friendly as possible to allow as many people as possible to play it, in turn expanding my scope of a player base.

Table

Description automatically generated

Figure

Based on these specs for Minecraft, one of the most optimised games of this era of gaming, I can assume the required specifications that a player would need to have would be lower than this. I think it’d be safe to assume that the Minimum requirements would be extremely low and therefore not worth putting down with recommended requirements possibly looking something like this. This would be because of the comparatively low memory and CPU load due to how advanced even 10-year-old hardware.

#### Hardware Requirements

CPU: Intel Core I3 – 3210 3.2 GHz / AMD A8 – 7600 APU 3.1 GHz

GPU (Integrated): Intel HD Graphics 4000 (Ivy Bridge) / AMD Radeon R5 series (Kaveri line) with OpenGL 4.4\*

GPU (Discrete): Nvidia GeForce 400 Series or AMD Radeon HD 7000 series with OpenGL 4.4

RAM: 4GB

### Limitations

My game does have some inherent limitations however that are a result of my choice of libraries and programming language. I can’t create a 3D game as well as work with different development platforms due to time constraints and possible library changes that would be required. I also can’t make a full game with multiple different dungeons along with hundreds of weapons for the same reasons.

# Design

## Overview

### Languages / Libraries

I will be making this using python as my language of choice in tandem with the pygame library to develop the video game itself thanks to its graphic and sound capabilities. I’m also utilising object-oriented programming for main aspects of the game concerning characters, weapons, and enemies as it allows for greater control over my code. I will also be using basic file management to store data for character statistics, item statistics, descriptions and more which would allow me to easily amend, update or drop data from the files and access specific information when I need to use them within the main program easily.

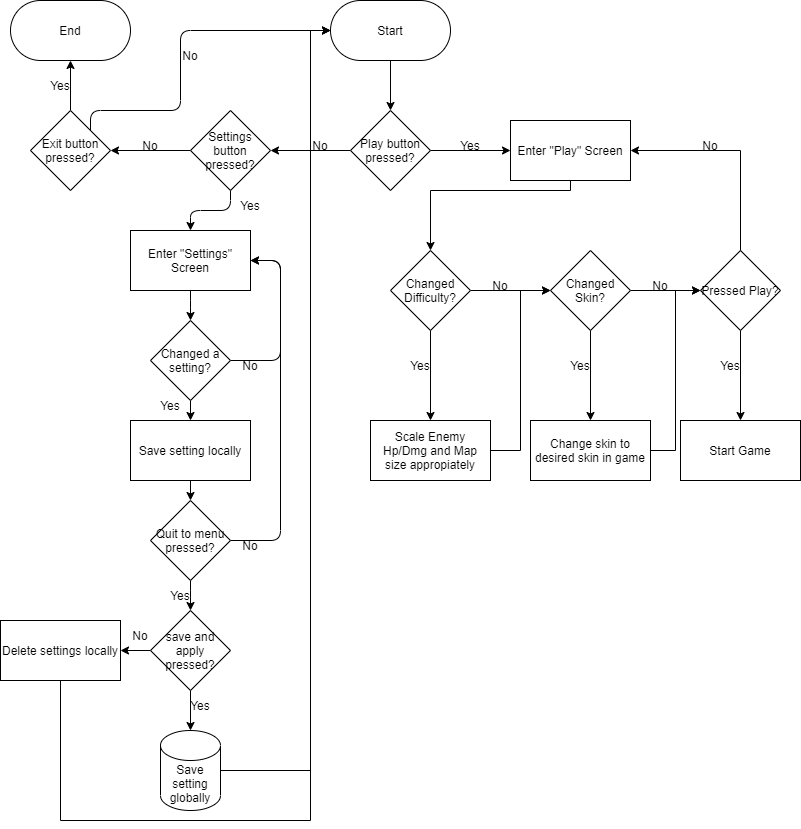
I’ll be using Visual Studio as my IDE to develop the video game as it has a robust debugging system which would allow me to streamline my workflow as I can search for where bugs occur and in turn find easy to fix bugs. It also has built-in access to GitHub so I can easily manage version control for my project which would minimise time lost if I make mistakes or want to go back to an earlier version of the project. It is also lightweight and easy to work with increasing my own productivity with intellisense being built in. I may also use Visual Studio Code as its extensions allow for more customization to fit my own style helping to minimize time wasted adjusting to a new system.

I feel like these should be sufficient for me to recreate the desired product I choose to create; however, I may end up using outside libraries and python libraries if necessary, during the technical solution such as Random which I would then add to this section when applicable. I’ll also make use the Math and library to work with Co-ordinate Matrices to manipulate my game objects on a technical scale as well as the OS library to manage file locations and negate any problems with running the game on some machines due to inaccessible files.

### Objective Flowchart

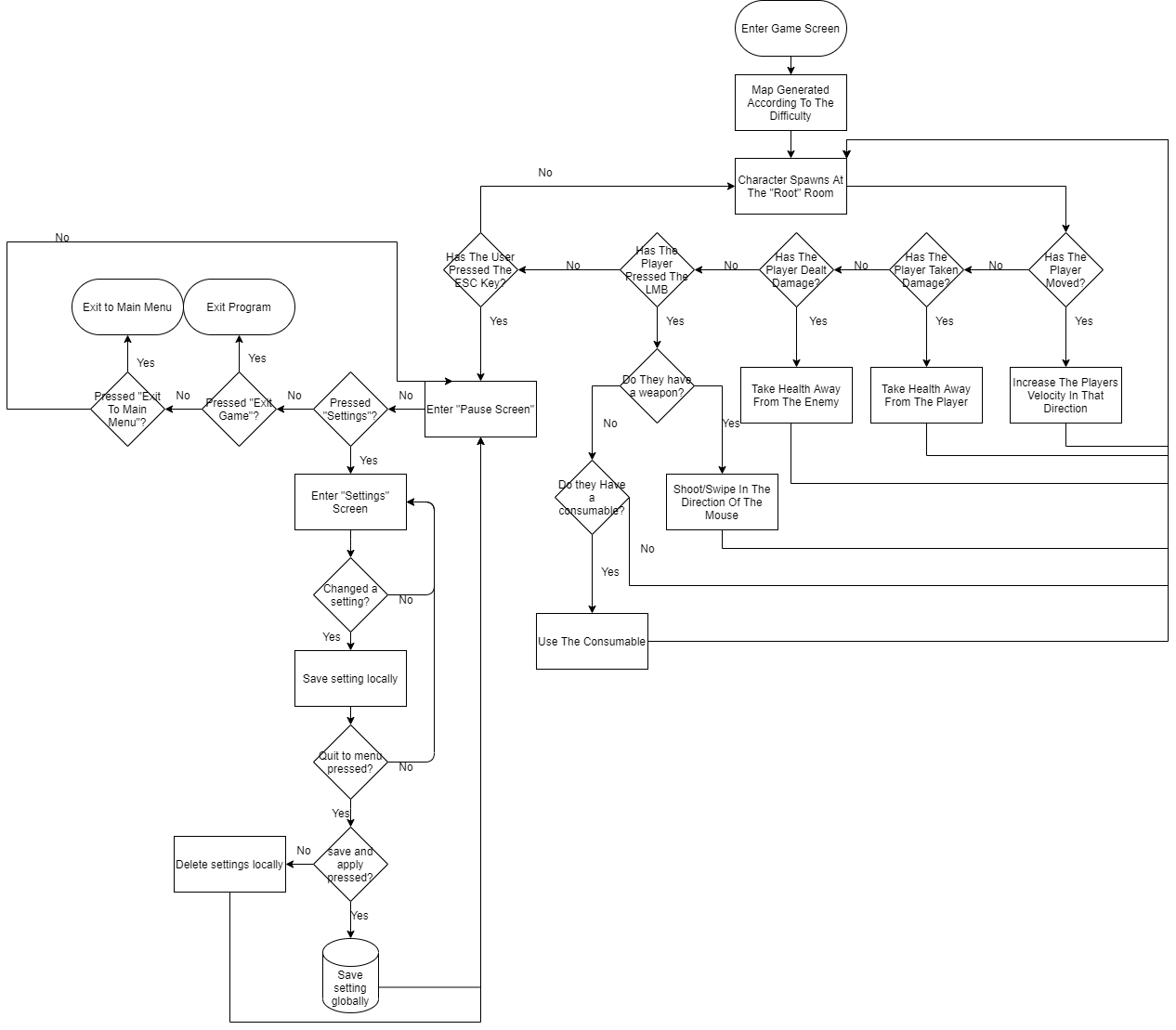
#### Main menu + Settings Screen + Play Screen

The main menu is designed to be a simple process of providing the user agency over what they want to change about their gameplay experience from general game settings to the player’s appearance and their desired difficulty. To achieve this would require some simple looping programs as evident on the following diagram with the “Main Menu”, “Settings Screen” and “Play Screen” having their own loops which continue indefinitely until a certain clause is met, for example the Play button being pressed for the “Main Menu” loop to end and the “Play Screen” loop to begin.



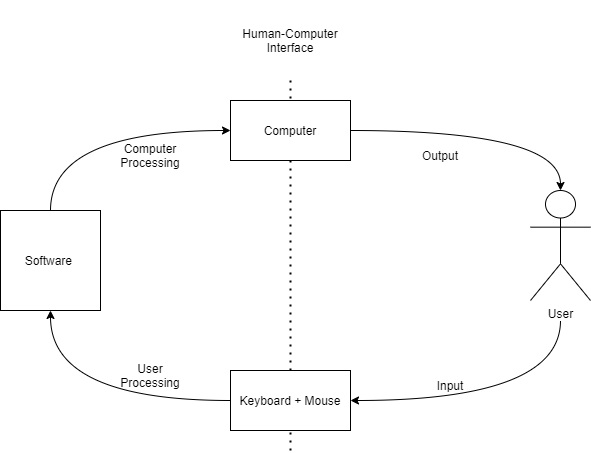
#### Game Screen

The Game Screen is a lot more complex with multiple loops for the main game itself although it I’ve generalised it to a few main ones in the flow chart below. It also has some from the “Settings Screen” and a new one for the “Pause Screen”. The large number of loops available means that I’ll have to make sure they don’t conflict with one another and that it doesn’t become fully single threaded as that could allow one bug to terminate the program while I’d only want that to occur for debugging allowing me to easily target bugs as I’d be able to tell where they originate from.



### Human Computer Interaction Diagram

The User and Hardware have a simple relationship in my project due to the inherent nature of it. The user outlines a set of objectives and goals they wish to complete in their head, and they then interact through computer peripherals, which in my case are a keyboard and mouse, by physically using them to complete these pre-defined tasks they have laid out subconsciously. This then introduces user inputs into the system which are converted in signals represented in in binary after being processed by internal algorithms. The software, aka the game, then takes these signals and uses them to carry out actions such as moving the character or accessing menus. The computer then processes through another internal algorithm to represent as an output on the monitor for the User to use to formulate a new set of goals or continue



## Classes & Functions

### Main Classes

#### Class Methods

### Class Diagram

### Main Functions

### Function Relationships

## Game processes

### “Main” method

The “Main” method would be the only method called outside of any other method and acts as the all-encompassing game process. This results in single threaded gameplay due to processes being handled in one place meaning that if something goes wrong then the code will stop working as intended. I will probably make use of the threading library to ensure that some processes like drawing would be acted out all at once allowing for smoother graphic display. A main method also allows for an easy way to manage and isolate problems as each function can be derived from multiple “update” and “draw” methods which are sorted into the “Main” method depending on what the game state is currently (which would be represented by an attribute of the “game” class)

### Spritesheets

A collage of a person

Description automatically generated with low confidence

Spritesheets are what is used most widely for game development when creating pixel art and as I am planning to use this, I need to create a class that will output a single sprite on a pygame surface based on a series of parameters inputted into the class when it is used. I will achieve this by having the x and y position of the sprite and the dimensions of the sprite I want to get which would then be used through a class that would blit this selection on a surface and return the surface acting as the desired sprite. This is faster and more memory efficient than storing each sprite as a file individually and loading each sprite as it this would add more time finding and allocating space to each sprite png.

### Vector Normalisation

Diagram

Description automatically generated

Vector Normalisation is a process in which when two cardinal directions are used at once, for character movement or for aiming weapons, the speed of movement is “normalised” to prevent any unintended speeds from taking place. I will use this by taking the x and y values of a change variable that will be act as the speed change and depending on if both the x and y change values are greater than one, it’d find the square root of the squares to get the magnitude of x and y needed to make speed equal in all directions.

This is incredibly useful and a prevalent process that is used in game development to help prevent games from feeling odd and uncomfortable to play and to also prevent non-uniform values and actions to avoid awkward bugs from appearing and unintentional exploits that could ruin the experience in a more fundamental manner.

### Collision Detection

Collision detection is a common practice in where to keep the game functioning as intended and avoid any unforeseen interactions between the character and background objects, all objects have a fundamental shape that they adhere to and an algorithm is run constantly where if 2 objects shapes (or more commonly known as hitboxes) are to intersect with each other then their positions are reset to before the collision. This allows for smooth and intuitive gameplay and doesn’t detach the player from the experience.

Specifically for collision between a player rectangle and a block rectangle there’d need to be specification between whether the player is moving in the x and y axis to allow for collision between the player and multiple blocks. If this system is not in place the player would frequently get in situations where they can’t move away from a block they collide with due to both axis being affected when colliding. This would then check if the player is moving in one direction or the other and assign the value of the sides colliding to one another. For player collision between the edges of the window a similar concept can be employed where if the x or y value of the player rectangle exceeds the width or height or is less than 0 on the x axis or on the y axis the corresponding value for the change variable will be made to be 0.

When it comes to collision with spells to the walls, we can just make a check to see if the rectangle of the spell collides with the block or exceeds the windows size and pop the bullet from the stack.

### Game States

As my game will have a UI system, I will need to work with game states to make sure the game runs smoothly and can switch between different screens easily. This will involve a game state that can be related to the Pause Screen, Main Menu Screen, Game Screen, Settings Screen, and Pre-Game Screen. These would all be used in the “Main” method in the “game” where a group of conditionals would be pertaining to each game state and result in a different set of processes being called dependant on this. I will need to tinker with the algorithms present in each so that some of them work regardless such as quitting the game and that the transition between game states isn’t jarring.

### Saving/Loading Data from Files

To allow for saving and loading of game data across the program, I will be loading data from files. I am going to make use of the Json library to load and dump data from existing text files and assign them to their own specific dictionaries making editing values easier as they would all be present in a file. I am also going to try and implement saving into my game which would just require the most important data to be stored and having a button which can read that scenario specific data such as the positions of the character, their items, and the map generation.

To avoid tampering of the data for the initial game and subsequent loaded games I could also employ an encryption system as well as the compression of game files to avoid any unintended changes that would ruin the game experience.

### Cooldowns and Delays

To create and handle these I will make use of a current time variable and a last updated time. The basic premise is using the current time variable, which is the amount of ticks that have passed,

### Animation

Animation is generally handled by having a current sprite variable, a current frame variable and a list of sprites derived through the spritesheet class. I’d use a delay to make the sprites change at a consistent time interval and so it is able to be seen by the user. Then every time the delay is over the current frame would have 1 added to it and you’d find the mod of it against the length of the list of sprites which would be the new current frame. This would increment through the list one by one acting as frames. This new current frame would be used to get the specific sprite from the list of sprites which is next in the list. Once this is done multiple times the sprite will change consistently over time according to the list of sprites.

## UI Processes

### Button

To make creating UI states easier I will be making a button class that will draw some text on the screen using a pre-defined font that is rendered within the class. If this text is hovered over the text will change colour and a method that will check if it is pressed and return a Boolean. It will accept values pertaining to its position on the screen and the text to be used on the button. This will allow me to implement buttons much faster than creating each button subsequently.

### Drawing Text

Drawing text is a much simpler process with only the rendering of the font and the creation of the text being needed, in which I have two different text classes for titles and text with different pre-defined fonts

## Character Processes

### MP Regen

By making use of the cooldowns explained before I delay MP regen for 0.05s and increment it by the value specified in constants.py

### Player Animation

Using the animation concept from before I then add a state on top of this which is changed depending on the direction the player is facing and whether they are moving. This is then used in a series of conditionals to discern which animation lists should be used.

## Enemies Processes

### Enemy AI

### Enemy 1

### Enemy 2

### Enemy 3

### Boss AI

## Map processes

### Floor Generation

### Room Generation

## Items

### Active Items

#### Spells

### Passive Items

# Technical Solution

## A/B/C Cross Reference

Single-dimensional arrays – 3.4.2, \_\_init\_\_ Method, Sprite Lists

Multi-dimensional arrays – 3.6.1

Dictionaries – 3.2.4

Simple OOP model – Entire Project

Simple user defined algorithms - 3.4.2, Animate Method

Generation of objects based on simple OOP model – 3.2.4

Simple mathematical calculations – 3.4.3, Math Method

Inheritance – 3.4.1 > 3.4.2

Records – 3.2.4.1

Recursive Algorithm – 3.2.4.1

## Main Game Programs

### Main.py

#### Game Class

class game:

def \_\_init\_\_(self):

pygame.init()

self.fps = 60

self.width = 1280

self.height = 720

self.rect = pygame.Rect(0, 0, self.width, self.height)

self.state = "game"

self.running = True

self.clock = pygame.time.Clock()

self.terrainSheet = pygame.image.load("Assets/terrain.png")

self.wizardSheet = pygame.image.load("Assets/wizard.png")

self.UIObj = UI(self)

def gameScale(self, baseMap):

self.tileWidth = self.width / len(baseMap[0])

self.tileHeight = self.height / len(baseMap)

def newGame(self):

self.window = pygame.display.set\_mode((self.width, self.height))

pygame.display.set\_caption("Mystic Maze")

self.scaleObj = scale(self)

self.scaleObj.tileScale(baseMap)

self.scaleObj.npcScale(baseMap)

self.tileWidth = self.scaleObj.tileWidth

self.tileHeight = self.scaleObj.tileHeight

self.npcWidth = self.scaleObj.npcWidth

self.npcHeight = self.scaleObj.npcHeight

self.roomObj = room(self)

self.roomObj.newRoom(baseMap)

self.playerObj = player(self, playerStats)

self.chaserObj = chaser(self, 600, 300)

self.spellObj = spell(self, straightSpell)

self.gameUIObj = gameUI(self)

def event(self):

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.running = False

def gameUpdate(self):

self.spellObj.update()

self.playerObj.update()

self.gameUIObj.update()

self.chaserObj.update()

def gameDraw(self):

self.window.fill(colour["white"])

self.roomObj.draw(baseMap)

self.spellObj.draw()

self.playerObj.draw()

self.chaserObj.draw()

self.gameUIObj.draw()

pygame.display.update()

def main(self):

self.newGame()

while self.running:

self.event()

self.clock.tick(self.fps)

self.keysPressed = pygame.key.get\_pressed()

self.mousePressed = pygame.mouse.get\_pressed()

self.mousePos = pygame.mouse.get\_pos()

if self.state == "main menu":

self.UIObj.mainMenuDraw()

#if self.state == "settings":

#if self.state == "pause":

if self.state == "game":

self.gameUpdate()

self.gameDraw()

pygame.quit()

sys.exit()

### Player.py

#### Player Class

class player:

def \_\_init\_\_(self, game, stats):

self.game = game

self.maxHp = stats["maxHp"]

self.hp = stats["hp"]

self.maxMp = stats["maxMp"]

self.mp = stats["mp"]

self.mpRegen = stats["mpRegen"]

self.atk = stats["atk"]

self.defe = stats["def"]

self.spd = stats["spd"]

self.baseSpd = 5

self.change = pygame.Vector2()

self.width = self.game.npcWidth

self.height = self.game.npcHeight

self.rect = pygame.Rect(0, 0, self.width, self.height)

self.rect.center = self.game.rect.center

self.spriteWidth = self.game.tileWidth

self.spriteHeight = self.game.tileHeight

self.spriteRect = pygame.Rect(self.rect.x, self.rect.y, self.spriteWidth, self.spriteHeight)

self.spritesheet = spritesheet(self.game.wizardSheet)

self.moveAnimCd = 150

self.idleAnimCd = 300

self.currentFrame = 0

self.animLastUpdated = 0

self.regenLastUpdated = 0

self.facing = "down"

self.rightIdleList = [pygame.image.load('Assets/Player Frames/Wizard05.png'),

pygame.image.load('Assets/Player Frames/Wizard06.png')]

self.downIdleList = [pygame.image.load('Assets/Player Frames/Wizard00.png'),

pygame.image.load('Assets/Player Frames/Wizard01.png')]

self.upIdleList = [pygame.image.load('Assets/Player Frames/Wizard10.png'),

pygame.image.load('Assets/Player Frames/Wizard11.png')]

self.rightList = [pygame.image.load('Assets/Player Frames/Wizard07.png'),

pygame.image.load('Assets/Player Frames/Wizard08.png'),

pygame.image.load('Assets/Player Frames/Wizard09.png'),

pygame.image.load('Assets/Player Frames/Wizard08.png')]

self.downList = [pygame.image.load('Assets/Player Frames/Wizard02.png'),

pygame.image.load('Assets/Player Frames/Wizard03.png'),

pygame.image.load('Assets/Player Frames/Wizard04.png'),

pygame.image.load('Assets/Player Frames/Wizard03.png')]

self.upList = [pygame.image.load('Assets/Player Frames/Wizard12.png'),

pygame.image.load('Assets/Player Frames/Wizard13.png'),

pygame.image.load('Assets/Player Frames/Wizard14.png'),

pygame.image.load('Assets/Player Frames/Wizard13.png')]

self.sprite = self.downIdleList[0]

def movement(self):

self.state = "idle"

self.change.x = 0

self.change.y = 0

self.vel = self.baseSpd \* self.spd

self.diagVel = (1 / math.sqrt(1\*\*2 + 1\*\*2)) \* self.vel

if self.game.keysPressed[pygame.K\_a]:

self.change.x = -self.vel

self.state = "left"

self.facing = "left"

if self.game.keysPressed[pygame.K\_d]:

self.change.x = self.vel

self.state = "right"

self.facing = "right"

if self.game.keysPressed[pygame.K\_s]:

self.change.y = self.vel

self.state = "down"

self.facing = "down"

if self.game.keysPressed[pygame.K\_w]:

self.change.y = -self.vel

self.state = "up"

self.facing = "up"

if self.change.x == self.vel and self.change.y == self.vel:

self.change.x = self.diagVel

self.change.y = self.diagVel

elif self.change.x == self.vel and self.change.y == -self.vel:

self.change.x = self.diagVel

self.change.y = -self.diagVel

elif self.change.x == -self.vel and self.change.y == self.vel:

self.change.x = -self.diagVel

self.change.y = self.diagVel

elif self.change.x == -self.vel and self.change.y == -self.vel:

self.change.x = -self.diagVel

self.change.y = -self.diagVel

self.windowCollision()

self.rect.x += self.change.x

self.axis = "x"

self.blockCollision()

self.rect.y += self.change.y

self.axis = "y"

self.blockCollision()

self.spriteRect.center = self.rect.center

def windowCollision(self):

if (self.rect.x + self.width + self.change.x) >= (self.game.width):

self.change.x = 0

if (self.rect.x + self.change.x) <= 0:

self.change.x = 0

if (self.rect.y + self.height + self.change.y) >= (self.game.height):

self.change.y = 0

if (self.rect.y + self.change.y) <= 0:

self.change.y = 0

def blockCollision(self):

if self.axis == "x":

for block in self.game.roomObj.blocks:

if pygame.Rect.colliderect(self.rect, block.rect):

if self.change.x < 0:

self.rect.left = block.rect.right

if self.change.x > 0:

self.rect.right = block.rect.left

if self.axis == "y":

for block in self.game.roomObj.blocks:

if pygame.Rect.colliderect(self.rect, block.rect):

if self.change.y > 0:

self.rect.bottom = block.rect.top

if self.change.y < 0:

self.rect.top = block.rect.bottom

def regeneration(self):

currentTime = pygame.time.get\_ticks()

if currentTime - self.regenLastUpdated > 50:

self.regenLastUpdated = currentTime

if self.mp != self.maxMp:

self.mp += self.mpRegen

def animation(self):

currentTime = pygame.time.get\_ticks()

if self.state == "idle":

if currentTime - self.animLastUpdated > self.idleAnimCd:

self.animLastUpdated = currentTime

self.currentFrame = (self.currentFrame + 1) % len(self.downIdleList)

if self.facing == "left":

self.sprite = pygame.transform.flip(self.rightIdleList[self.currentFrame], True, False)

if self.facing == "right":

self.sprite = self.rightIdleList[self.currentFrame]

if self.facing == "down":

self.sprite = self.downIdleList[self.currentFrame]

if self.facing == "up":

self.sprite = self.upIdleList[self.currentFrame]

else:

if currentTime - self.animLastUpdated > self.moveAnimCd:

self.animLastUpdated = currentTime

self.currentFrame = (self.currentFrame + 1) % len(self.downList)

if self.state == "left":

self.sprite = pygame.transform.flip(self.rightList[self.currentFrame], True, False)

if self.state == "right":

self.sprite = self.rightList[self.currentFrame]

if self.state == "down":

self.sprite = self.downList[self.currentFrame]

if self.state == "up":

self.sprite = self.upList[self.currentFrame]

def update(self):

self.movement()

self.animation()

self.regeneration()

def draw(self):

#pygame.draw.rect(self.game.window, colour["blue"], self.spriteRect)

#pygame.draw.rect(self.game.window, colour["red"], self.rect) self.game.window.blit(pygame.transform.scale(self.sprite, (self.spriteWidth, self.spriteHeight)), self.spriteRect)

### Sprites.py

#### Terrain Class

class terrain:

def \_\_init\_\_(self, game, x, y):

self.game = game

self.x = x \* self.game.tileWidth

self.y = y \* self.game.tileHeight

self.width = self.game.tileWidth

self.height = self.game.tileHeight

self.spriteWidth = spriteSize

self.spriteHeight = spriteSize

self.rect = pygame.Rect(self.x, self.y, self.width, self.height)

self.spritesheet = spritesheet(self.game.terrainSheet)

#### Block Class

class block(terrain):

def \_\_init\_\_(self, game, x, y):

super().\_\_init\_\_(game, x, y)

self.sprite = self.spritesheet.getSprite(192, 608, self.spriteWidth, self.spriteHeight, colour["black"])

def draw(self):

self.game.window.blit(pygame.transform.scale(self.sprite, (self.width, self.height)), (self.rect.x, self.rect.y))

#### Floor Class

class floor(terrain):

def \_\_init\_\_(self, game, x, y):

super().\_\_init\_\_(game, x, y)

self.sprite = self.spritesheet.getSprite(416, 96, self.spriteWidth, self.spriteHeight, colour["black"])

def draw(self):

self.game.window.blit(pygame.transform.scale(self.sprite, (self.width, self.height)), (self.rect.x, self.rect.y))

### Maps.py

#### Map Class

class map(room):

def \_\_init\_\_(self):

self.width = 5

self.height = 5

self.numFloors = 1

self.numRooms = 10

self.floorTileMap = []

self.roomData = {}

def newMap(self):

self.floorList()

self.counter = 1

self.generate()

print(self.floorTileMap)

def generate(self):

if self.counter != self.numRooms:

self.Room1 = self.Room2

directionDict = {"left":False, "right":False, "up":False, "down":False}

directionRandom = random.randint(1,4)

if directionRandom == 1:

directionDict["left"] = True

if directionRandom == 2:

directionDict["right"] = True

if directionRandom == 3:

directionDict["up"] = True

if directionRandom == 4:

directionDict["down"] = True

self.roomData[f"room{self.counter}"] = [directionDict]

self.counter += 1

self.generate()

def lists(self):

self.floor1D = [0 for x in range(self.width)]

for i in range(self.floorHeight):

self.floorTileMap.append(self.floor1D)

self.Room2 = self.floorTileMap

#### Room Class

class room:

def \_\_init\_\_(self, game):

self.game = game

self.currentRoom = 0

def newRoom(self, tilemap):

self.blocks = []

for y, row in enumerate(tilemap):

for x, tile in enumerate(row):

if tile == "B":

b = block(self.game, x, y)

self.blocks.append(b)

def draw(self, tilemap):

for y, row in enumerate(tilemap):

for x, tile in enumerate(row):

if tile == "B":

b = block(self.game, x, y)

b.draw()

if tile == "F":

f = floor(self.game, x, y)

f.draw()

### UI.py

#### UI Class

class UI:

def \_\_init\_\_(self, game):

self.game = game

def mainMenuUpdate(self):

return

def mainMenuDraw(self):

button(self.game, "prstart.ttf", 32, "test", self.game.rect.center[0], self.game.rect.center[1])

return

def pauseMenuUpdate(self):

return

def pauseMenuDraw(self):

return

def settingsUpdate(self):

return

def settingsDraw(self):

return

#### GameUI Class

class gameUI:

def \_\_init\_\_(self, game):

self.game = game

self.hpSize = 5

self.hpX = 15

self.hpY = 15

self.hpHeight = 25

self.mpSize = 4

self.mpX = 25

self.mpY = 40

self.mpHeight = 20

self.itemBoxesY = 600

self.itemBoxesWidth = 100

self.itemBoxesHeight = 100

def barUpdate(self):

self.hpWidth = (100 \* (self.game.playerObj.hp / self.game.playerObj.maxHp)) \* self.hpSize

self.hpBackWidth = 100 \* self.hpSize

self.mpWidth = (100 \* (self.game.playerObj.mp / self.game.playerObj.maxMp)) \* self.mpSize

self.mpBackWidth = 100 \* self.mpSize

self.hpRect = pygame.Rect(self.hpX, self.hpY, self.hpWidth, self.hpHeight)

self.hpBackRect = pygame.Rect(self.hpX, self.hpY, self.hpBackWidth, self.hpHeight)

self.mpRect = pygame.Rect(self.mpX, self.mpY, self.mpWidth, self.mpHeight)

self.mpBackRect = pygame.Rect(self.mpX, self.mpY, self.mpBackWidth, self.mpHeight)

def barDraw(self):

pygame.draw.rect(self.game.window, colour["black"], self.hpBackRect)

pygame.draw.rect(self.game.window, colour["red"], self.hpRect)

pygame.draw.rect(self.game.window, colour["black"], self.mpBackRect)

pygame.draw.rect(self.game.window, colour["blue"], self.mpRect)

def itemBoxesDraw(self):

self.itemBoxesX = 600

for i in range(5):

self.itemBoxesX += 110

self.itemBoxesRect = pygame.Rect(self.itemBoxesX, self.itemBoxesY, self.itemBoxesWidth, self.itemBoxesHeight)

pygame.draw.rect(self.game.window, colour["brown"], self.itemBoxesRect)

def update(self):

self.barUpdate()

def draw(self):

self.itemBoxesDraw()

self.barDraw()

### Item.py

#### Spell Class

class spell:

def \_\_init\_\_(self, game, stats):

self.game = game

self.type = stats["type"]

self.dmg = stats["dmg"]

self.numShots = stats["numShots"]

self.limit = stats["limit"]

self.spd = stats["spd"]

self.size = stats["size"]

self.mpCost = stats["mpCost"]

self.cooldown = stats["cooldown"]

self.lastUpdated = 0

self.projList = []

def move(self):

for index, bullet in enumerate(self.projList):

for block in self.game.roomObj.blocks:

collide = pygame.Rect.collidepoint(block.rect, (bullet[0], bullet[1]))

if collide == True:

self.projList.pop(index)

if bullet[0] >= self.game.width or bullet[0] <= 0:

self.projList.pop(index)

if bullet[1] >= self.game.height or bullet[1] <= 0:

self.projList.pop(index)

bullet[0] += bullet[2]

bullet[1] += bullet[3]

def straightSpawn(self):

if self.game.mousePressed[0]:

if self.game.playerObj.mp >= self.mpCost:

currentTime = pygame.time.get\_ticks()

if currentTime - self.lastUpdated > self.cooldown:

self.lastUpdated = currentTime

mouseX, mouseY = self.game.mousePos

x = self.game.playerObj.rect.center[0]

y = self.game.playerObj.rect.center[1]

distanceX = mouseX - x

distanceY = mouseY - y

angle = math.atan2(distanceY, distanceX)

velX = self.spd \* math.cos(angle)

velY = self.spd \* math.sin(angle)

self.projList.append([x, y, velX, velY])

self.game.playerObj.mp += -self.mpCost

def shotgunSpawn(self):

return

def update(self):

self.move()

if self.type == "straight":

self.straightSpawn()

def draw(self):

for bullet in self.projList:

posX = int(bullet[0])

posY = int(bullet[1])

pygame.draw.circle(self.game.window, colour["orange"], (posX, posY), (self.size \* (5/3)))

pygame.draw.circle(self.game.window, colour["red"], (posX, posY), self.size)

### Enemy.py

#### Enemy Class

class enemy:

def \_\_init\_\_(self, game, x, y):

self.game = game

self.x = x

self.y = y

self.width = self.game.tileWidth

self.height = self.game.tileHeight

self.rect = pygame.Rect(self.x, self.y, self.width, self.height)

self.spriteWidth = self.game.tileWidth

self.spriteHeight = self.game.tileHeight

self.spriteRect = pygame.Rect(self.rect.x, self.rect.y, self.spriteWidth, self.spriteHeight)

def draw(self):

self.game.window.blit(self.sprite, self.rect)

#### Turret Class

class turret(enemy):

def \_\_init\_\_(self, game, x, y):

super().\_\_init\_\_(game, x, y)

self.maxHp = 100

self.hp = 100

self.atk = 1

self.spd = 1

self.dmg = 1

self.bulletSize = 10

self.bulletSpd = 5

self.projList = []

self.atk1Cooldown = 750

self.atk1LastUpdated = 0

def attack1(self):

currentTime = pygame.time.get\_ticks()

if currentTime - self.atk1LastUpdated > self.atk1Cooldown:

self.atk1LastUpdated = currentTime

playerX = self.game.playerObj.rect.center[0]

playerY = self.game.playerObj.rect.center[1]

x = self.rect.center[0]

y = self.rect.center[1]

distanceX = playerX - x

distanceY = playerY - y

angle = math.atan2(distanceY, distanceX)

velX = self.bulletSpd \* math.cos(angle)

velY = self.bulletSpd \* math.sin(angle)

self.projList.append([x, y, velX, velY])

def moveBullet(self):

for index, bullet in enumerate(self.projList):

for block in self.game.roomObj.blocks:

collide = pygame.Rect.collidepoint(block.rect, (bullet[0], bullet[1]))

if collide == True:

self.projList.pop(index)

if bullet[0] >= self.game.width or bullet[0] <= 0:

self.projList.pop(index)

if bullet[1] >= self.game.height or bullet[1] <= 0:

self.projList.pop(index)

bullet[0] += bullet[2]

bullet[1] += bullet[3]

def update(self):

self.attack1()

self.moveBullet()

def draw(self):

pygame.draw.rect(self.game.window, colour["blue"], self.rect)

for bullet in self.projList:

posX = int(bullet[0])

posY = int(bullet[1])

pygame.draw.circle(self.game.window, colour["orange"], (posX, posY), (self.bulletSize \* (5/3)))

pygame.draw.circle(self.game.window, colour["red"], (posX, posY), self.bulletSize)

### Framework.py

#### Spritesheet Class

class spritesheet:

def \_\_init\_\_(self, sheet):

self.sheet = sheet

def getSprite(self, x, y, width, height, colour):

sprite = pygame.Surface((width, height))

sprite.blit(self.sheet, (0, 0), (x, y, width, height))

pygame.transform.scale(sprite, (width, height))

sprite.set\_colorkey(colour)

return sprite

#### Button Class

class button:

def \_\_init\_\_(self, game, font, size, text, x, y):

self.game = game

self.font = pygame.font.Font(font, size)

self.text = self.font.render(text, True, colour["gold"])

self.hoverText = self.font.render(text, True, colour["cream"])

self.rect = self.text.get\_rect()

self.rect.x = x

self.rect.y = y

self.rect.center = (x + self.rect.width/2, y + self.rect.height/2)

self.game.window.blit(self.text, (x, y))

def pressed(self):

if self.rect.collidepoint(self.game.mousePos):

if self.game.mousePressed[0]:

return True

return False

return False

#### DrawText Class

class drawText:

def \_\_init\_\_(self, game, x, y, font, size, text, colour):

self.game = game

self.font = pygame.font.Font(font, size)

self.rect = self.font.get\_rect()

self.text = self.render(text, True, colour)

self.game.window.blit(text, (x, y))

#### Scale Class

class scale:

def \_\_init\_\_(self, game):

self.game = game

def tileScale(self, baseMap):

self.tileWidth = self.game.width / len(baseMap[0])

self.tileHeight = self.game.height / len(baseMap)

def npcScale(self, baseMap):

self.npcWidth = (self.game.width / len(baseMap[0])) \* 0.6

self.npcHeight = (self.game.height / len(baseMap)) \* 0.6

### Constants.py

spriteSize = 32

playerStats = {"maxHp" : 100,

"hp" : 100,

"maxMp" : 100,

"mp" : 100,

"mpRegen" : 0.5,

"atk" : 1,

"def" : 1,

"spd" : 1}

straightSpell = {"type" : "straight",

"dmg" : 1,

"numShots" : 1,

"limit" : 10,

"spd" : 7.5,

"size" : 5,

"mpCost" : 10,

"cooldown" : 300}

shotgunSpell = {"type" : "straight",

"dmg" : 1,

"numShots" : 5,

"limit" : 10,

"spd" : 7.5,

"size" : 5,

"mpCost" : 10,

"cooldown" : 300}

opSpell = {"type" : "straight",

"dmg" : 1000,

"numShots" : 10,

"limit" : 1000000,

"spd" : 25,

"size" : 15,

"mpCost" : 0,

"cooldown" : 1}

colour = {"white" : (255, 255, 255),

"black" : (0, 0, 0),

"grey" : (128, 128, 128),

"brown" : (102, 51, 0),

"light brown" : (153, 102, 0),

"red" : (255, 0, 0),

"green" : (0, 255, 0),

"blue" : (0, 0, 255),

"yellow" : (255, 255, 0),

"cream" : (255, 204, 102),

"gold" : (204, 153, 0),

"orange" : (255, 102, 0),

"purple" : (102, 0, 102)}

baseMap = ["BBBBBBBFFBBBBBBB",

"BFFFFFFFFFFFFFFB",

"BFFFFFFFFFFFFFFB",

"FFFFFFFFFFFFFFFF",

"FFFFFFFFFFFFFFFF",

"FFFFFFFFFFFFFFFF",

"BFFFFFFFFFFFFFFB",

"BFFFFFFFFFFFFFFB",

"BBBBBBBFFBBBBBBB"]

# Testing