NEA - The Maze Game

Hugo Whittome

September 2020

Contents

1 Analysis

1.1 Overview

This is an exploration game, where you explore a randomised maze collecting items and followers on your way to help defeat the monsters, while also trying to find the escape route - which will be a room with a trapdoor.

1.2 Maze Generation

1.2.1 Maze Needed

My plan for the maze is for it for it to be infinite, meaning that it only generates part of the maze at a time, and as you explore, you uncover more of the maze. However, for memory efficiency, the maze that is no longer loaded, won't be stored in memory and so deleted.

1.2.2 Types

There is both labyrinths and mazes. Labyrinths have only one path. This means that there is minimal choice in where the user can decide to go. The other type is mazes. These are multicursal, meaning it has multiple paths. This allows the user to choose their own path.

1.2.3 Approaches to Generation

- Cellular Automation Algorithm

  This is based on John Conway's Game of Life, where a cell is created if it has exactly 3 neighbours and can survive it has 1-5 neighbors. However, this means that with the same starting pattern, the same maze will be created everytime.

- Prim's Algorithm

  This is where a random point on the maze is chosen as the starting point. Then all the surrounding areas are added to a list. Then the program continually generates new sections and adds more areas to the list, until the list is completely empty and all the spaces on the board is taken up. The positives with this is that it creates a randomised maze everytime, that takes up the whole map. However, the disadvantage is that when generating more sections, the maze cannot go back on itself.

1.2.4 Conclusion

In conclusion, to make an infinite maze, I shall be using my own algorithm. This is slightly based off Prim's Algorithm, however instead of filling up the whole board, it leaves gaps. This is done by randomising entrances when placing a cell and then added only those possibilities to the list to generate more. This means that when the player moves north, it is able to generate paths that go back on itself however lead to dead end and not connect back up to the maze. This I feel will make a more dynamic maze when continually exploring the maze.

1.3 Existing Solutions

1.3.1 The Binding of Isaac

A similar game is The Binding of Isaac. What I liked about this game was the exploration and randomness, along with the challenge of fighting monsters in different rooms. However, I found it frustrating that there was limiting exploration on each level, as the level is not infinite. Furthermore, another issue I had was the only help that you could get was a familiar, in my game I wish to improve this by having set NPCs that you can find when exploring each level. Also, in binding of Isaac, the scrolling is not smooth, jumping between each room. Also there are no corridors, making it seem less maze like.

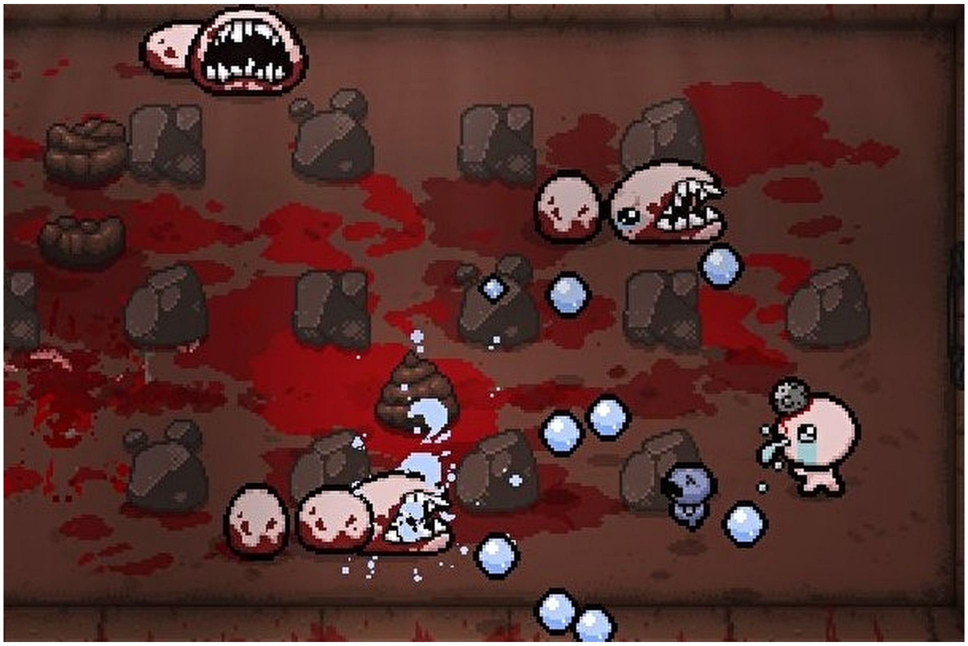


Figure 1: Picture from the binding of Isaac, showing the player (bottom right) attacking the enemies

1.4 End Users

1.4.1 Description

Teenagers who enjoy exploration video games.

1.4.2 Questionnaire

- Have you played an exploration game before?

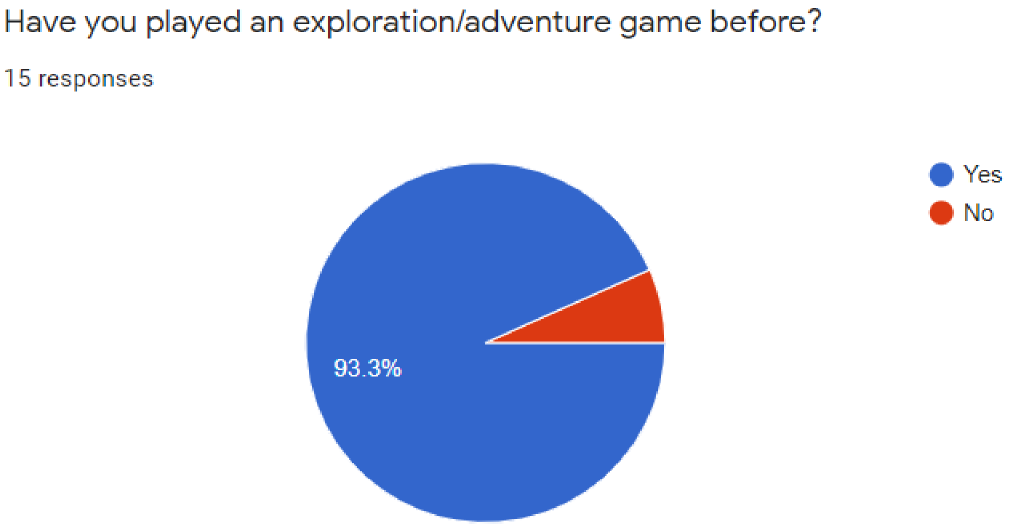


Figure 2: Responses from survey showing most people have played an exploration game

- How important is each section when looking for a game?
    - Boss fights
    - NPCs that you can interact with
    - Enemies that attack you
    - Good story

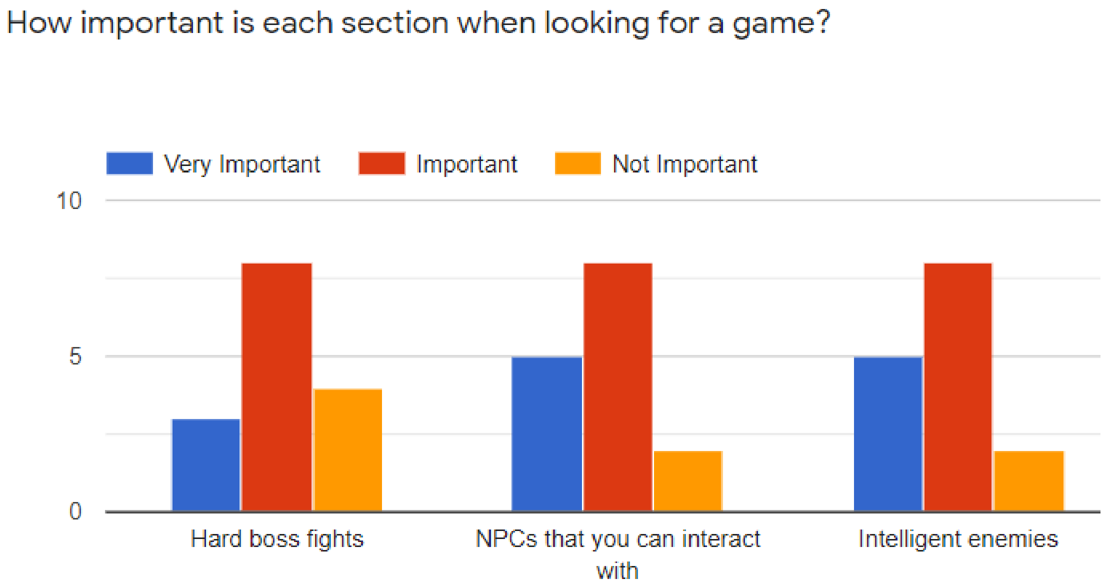


Figure 3: Responses from the story showing that it is equally important to have intelligent enemies and NPCs you can interact with

- What era do you like games to be designed as?
    - Future
    - Modern
    - Medieval
    - Stone Age
    - Multiple eras

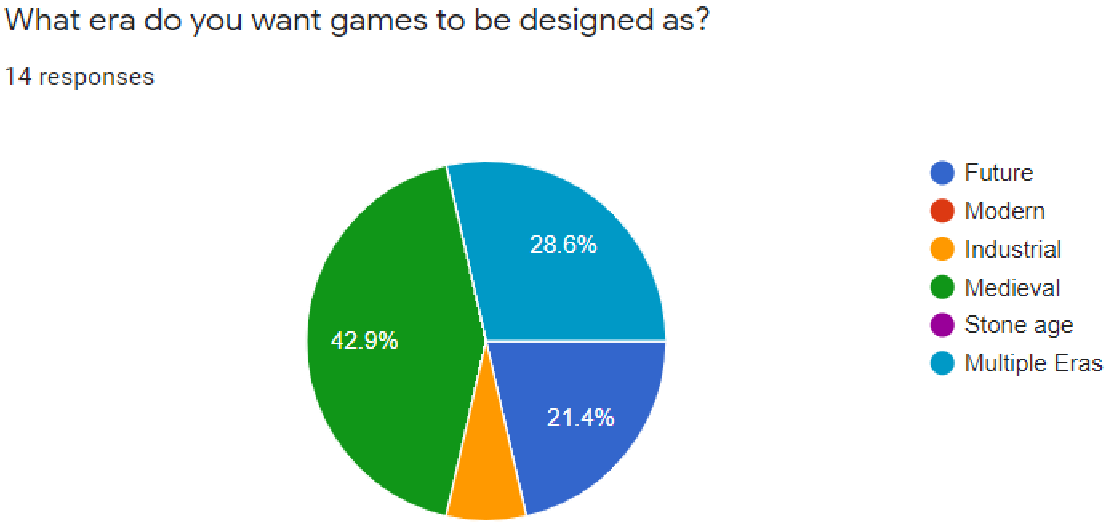


Figure 4: Responses from survey showing that most people like a Medieval design

- Which do you prefer a weight-based system for the inventory (e.g. in Skyrim) or a space-based system (e.g. Minecraft)?

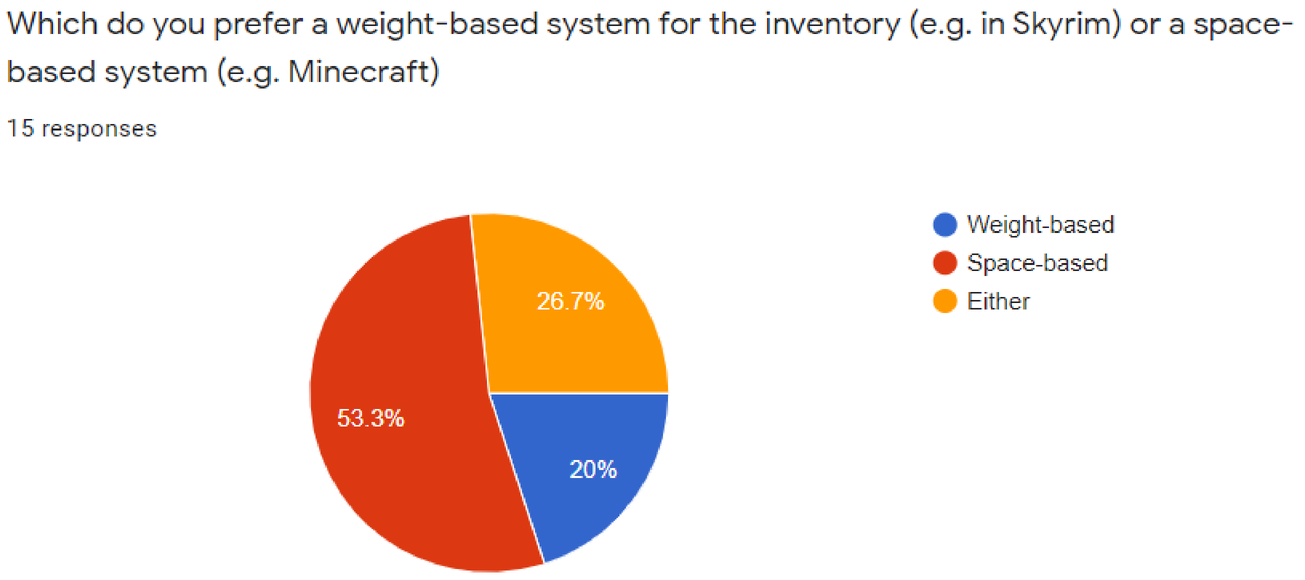


Figure 5: Responses from survey showing that most people like a weight-based system in a game

- Do you prefer being able to move while attacking or a Pokémon style attack system?

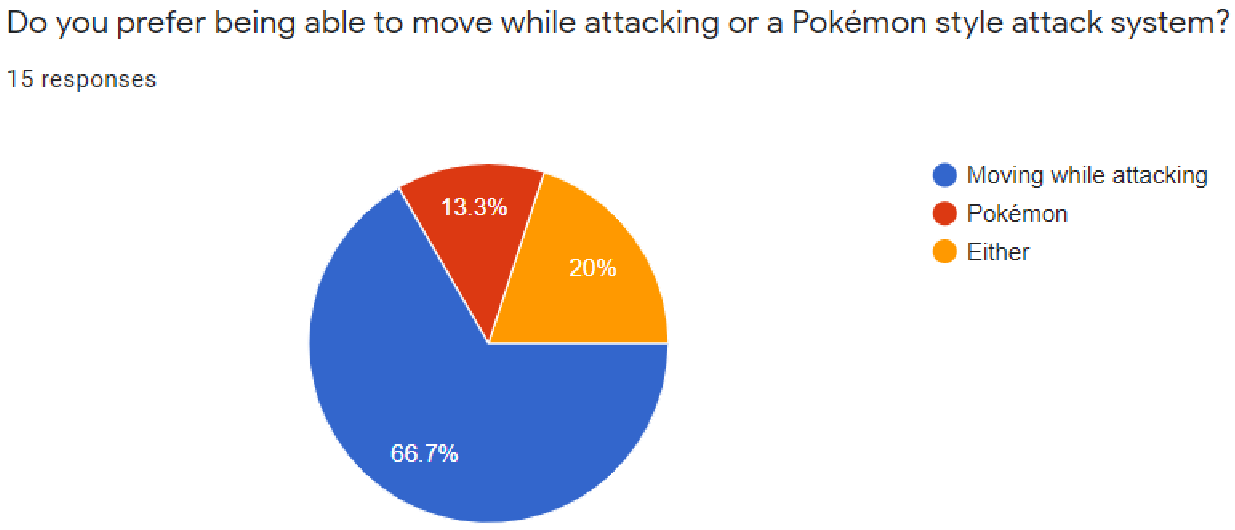


Figure 6: Responses from survey showing that the attacking system should allow you to still control the player

- Have you played "The Binding of Isaac"?

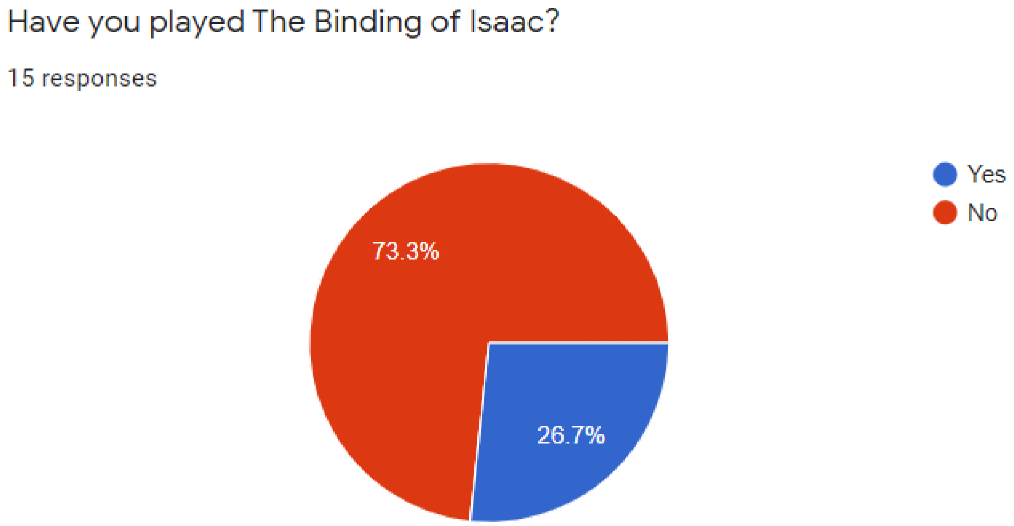


Figure 7: Responses from survey showing that most people in this survey had not played The Binding of Isaac

  - If they had I asked what they liked about the game and what they think could be better (talked about in the conclusion)

1.4.3 Conclusion

As shown in Figure 2, most people taking the survey had played and exploration or adventure game before, this means that the survey would be somewhat respective of the audience this game will be made for. Surprisingly, in Figure 3, most people believed that hard boss fights are less important than NPCs you can interact with and intelligent enemies, so I will prioritise those features over creating boss fights. For the design, I will be going for a Medieval theme as it seems as though most people prefer that design scheme (as shown in Figure 4). Also, I have chosen to use a space-based system for the inventory as in Figure 5, most people responding that they prefer that system. Also, I will allow the player to attack at any point in the maze (for example shooting a projectile) because (as shown in Figure 6) people prefer it over an attacking GUI.

As shown in Figure 7, only a few percentage had played The Binding of Isaac. when asked what they liked about the game, the responses included:

- The rougelike aspect (a subgenre of game that have generated levels, and tile-based graphics)
- The replayability and unique style

Most responses where talking about the rougelike aspect or the randomised levels. From this I have decided to use tile-based graphics with most sprites with a resolution of 64 pixels by 64 pixels to make it more rougelike also I want to make the maze as randomised as possible, meaning that many stats will be randomised.

1.5 Objectives

- Have an effective rendering system
  - This system must use OpenGL - as it is the graphics library I am using for this project.
  - This means having a system in place where I can call a function, giving it a set of values, and then it will be automatically rendered, so that I do not have to deal with keeping track of how much of the buffer is used up
  - So once this is complete, I should have be able to render a tile, or multiple tiles on the screen.
  - Create a camera class that can move in the 2D plane with the keyboard.
- Be able to generate an infinite maze
  - This needs the rendering system to be finished, so that I can render the maze once generated.
  - The maze needs to be stored in effective means that means that it will not slow down everytime it generates more of the maze.
  - This needs to be able to generate a maze from nothing, with most of the board filled up.
  - Once this is in place, I can then make is so that once you can move in each direction and the maze will generate more of itself.
  - Create a class for the player, allowing them to be rendered into the maze and have the camera follow the player.
- Create NPCs that rome the maze and can start following you
  - Create an NPC class that can be placed and rendered into the world.
  - Make it so that the follower can ask the level for the shortest route (which will use the A* algorithm).
  - Make it so that once they have the direction they need to go in, that they can move around the map.
  - Add different types of followers.
- Add items and a way to collect them with a simple space-based inventory system
  - Add an inventory to each mob (player, follower, enemy)
  - Create an item class that can be found in the maze.
  - Allow the item to be picked up and put inside the inventory of the player
  - Make an inventory system, so that if the player has too much in their inventory they can chose what to get rid of.
  - Allow items to be parsed to the followers (so that they act like storage for the player)
- Add combat into the game
  - Add projectile class that can be created by the player and rendered onto the map.
  - Allow the projectile to move in the direction the player is facing, and when it collides with an entity or solid tile, it will delete itself.
  - Add a particle system, so that the projectile will produce particles with a solid colour, that will decay over time.
  - Adapt NPC class to allow for attacking (enemies will be NPCs that are attacking the player).
  - Add a health and other stats (strength, agility ...) for every mob (player, follower, enemy).
  - Make it so that when a projectile hits an entity it deals a random amount of damage (in a given range), and check if the entity has died or not. Create a system to deal with the player's death.
  - Create an algorithm for the enemies to attack the player and their followers, also allow the followers to use the same algorithm to attack the enemy
  - Allow the enemies to have followers, who also attack the player and their followers.
  - Add multiple weapons, which have different damages and particle effects.
- Create different rooms that can be found in the maze.
  - Add multiple different rooms the player can find while exploring the maze.
  - This will include creating a chest that randomly generates items inside, which the player can pick up.
  - Update the maze generation slightly so that it can randomly generate the different types of rooms.
- Create a menu system
  - Create a layer for handling GUI objects.
  - Create button objects that can run a function when clicked.
  - Overhaul the inventory menu to work with the new system.

  - Make the game updates to be paused when inside a menu.
  - Create a main menu, where you can start a new game.
  - Allow the user to get back to the main menu while playing the game.

- Finalise everything
  - Add more stats to the mobs, which results in different effects to your damage and the number of followers you can have.
  - Add food into the game which acts like a potion and have random sprites generated.
  - Clean up anything left unfinished and get rid of any bugs left over.
  - Update the sprites of the maze so that it has a medieval theme.

2 Documented Design

2.1 Overview

As I am creating a game, it must be designed with that in mind. Therefore, most parts of the game will be stored in classes, with a couple of singletons designed with doing the key aspects of the game, e.g. managing the application, rendering the game, and logging warnings or messages so that it is easier to debug errors. Furthermore, the game must be able to be compiled for both Linux and Windows as I use both Linux and Windows. Finally when developing the game, it will be compiled with some features that will allow me to alter settings while inside the game, which will allow for faster development.

Furthermore, during my development of the project I shall need to use some packages to allow me to more easily create the game, without having to program every little part of the game, the packages I will be using are:

- FreeType - This is used to make text rendering easier.
- GLEW and GLFW - These are used so as an OpenGL implementation I can use to render objects on the screen.
- ImGui - This is used for when debugging - and is not compiled in the release version. It allows for me to have a simple menu which I can program to change variables or call functions when th program is running.
- GLM - This is a simple library designed for OpenGL. It has a few utils that are easy to use and are extremely optimised.
- stb_image - This is for loading images into the game for textures.

2.2 Maze Generation

2.2.1 Storage

I have developed a special method of storing the rooms in the maze, to start off, it will be stored in a 1d array (which acts like a 2d array when getting rooms, using coordinates), then when generating more of the maze, when the player moves, 2 variables will be altered storing the offset of the x and y coordinates - which will allow the maze to not move and reallocate all the rooms stored.

Instead imagine when storing a room at position (0, 0) or index 0 in the array. When the maze needs to generate more of the maze on the north, the position in the array of that room will not change, it will still be at index 0, however the coordinates will change as its moved down therefore its now at (0, -1), and the new rooms generated will be at the end of the maze. However when accessing the array with the coordinate system it will seem as though all the rooms have moved down 1. This will save processing power moving all the rooms to different indices in the array.

However, the rooms also will have to store coordinates of where they are. This results in having to go through all the rooms and updating the coordinates. To reduce this as much as possible, the coordinates will only be updates when one of the offsets loops back to the center, so for example going back to 0.

2.2.2 Prototype

For a prototype of the generation, I decided to write it in python with a room just consisting of being a cross section, this was to make sure that it wasn't too complex, while keeping the basic idea of the generation.

The figure above briefly shows planning behind how the maze generation works, with the rooms outlined in black, as rooms that have a set place, with then the rooms highlighted in blue being the rooms yet to be generated, and thus in the "current" list.

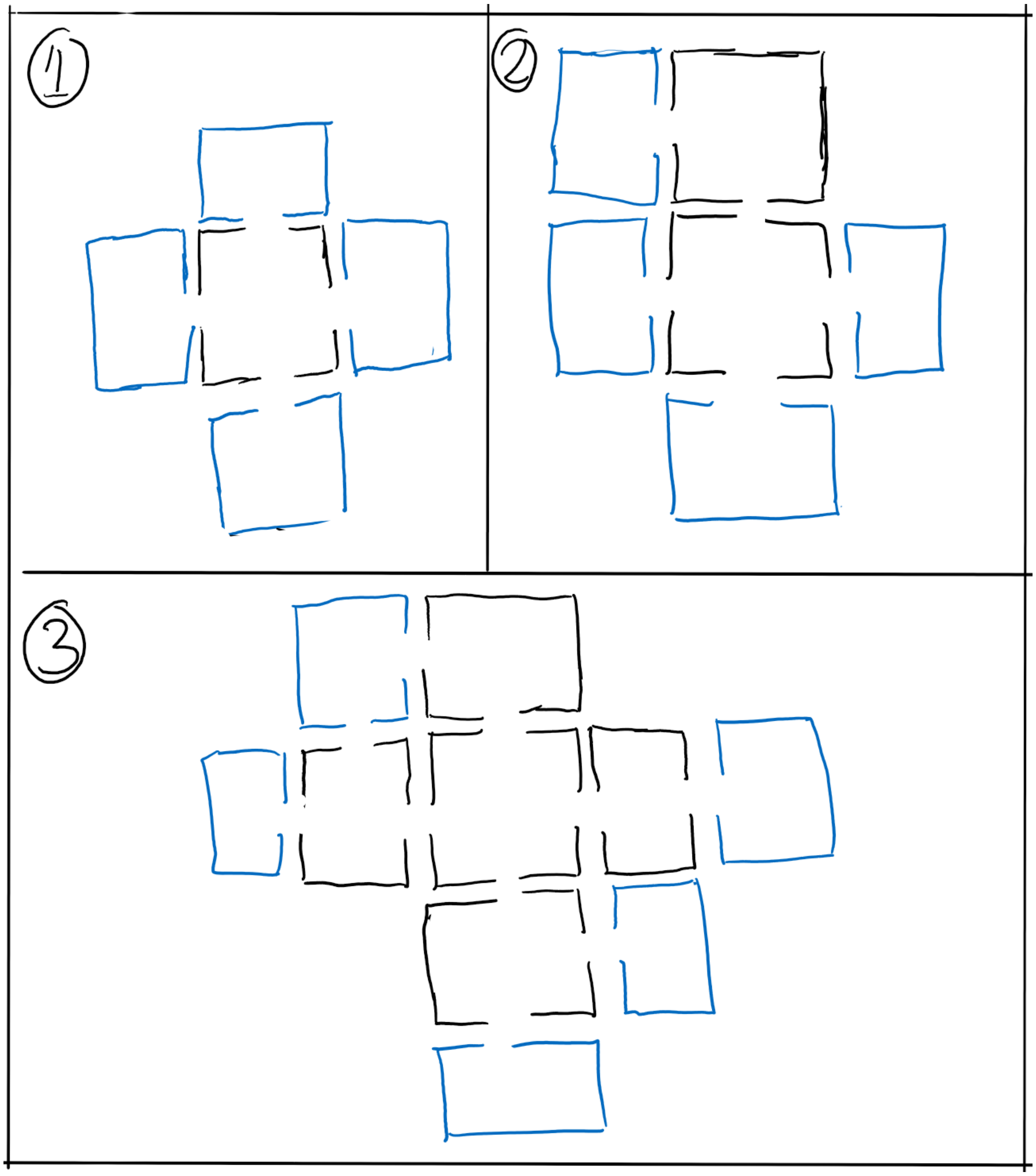


Figure 8: Steps followed by the maze generation

```python
from random import randint

class Sector: # This is a simple sector that represents a room in the game
    #             North  South  East   West
    enterences = [False, False, False, False] # Stores which entrances are open

    def __init__(self, north, south, east, west): # Simple initialiser
        self.enterences = [north, south, east, west]

    def __repr__(self): # Returns a string representation of the room, using 'X' to represent walls
        printStr = ""
        if self.enterences[0]:
            printStr += "X X\n"
        else:
            printStr += "XXX\n"
        if self.enterences[3]:
            printStr += "  "
        else:
            printStr += "X "
        if self.enterences[2]:
            printStr += " \n"
        else:
            printStr += "X\n"
        if self.enterences[1]:
            printStr += "X X\n"
        else:
            printStr += "XXX\n"
        return printStr

    # These functions just switch the each entrance from being open to close and close to open
    def switchNorth(self):
        self.enterences[0] = not self.enterences[0]

    def switchSouth(self):
        self.enterences[1] = not self.enterences[1]

    def switchEast(self):
        self.enterences[2] = not self.enterences[2]

    def switchWest(self):
        self.enterences[3] = not self.enterences[3]

    def __eq__(self, o):
        if not isinstance(o, Sector):
            return False
        return True

    def __ne__(self, o):
        return not self == o

def printHugeString(theHugeString): #   As the program produces a string that is normally too long to print
    lines = theHugeString.split("\n") # This print function splits it up into chunks that can be printed
    for i in range(0, len(lines), 10):
        print("\n".join(lines[i: i + 10]))
    #if len(lines) % 10 != 0:
    #    print("\n".join(lines[(len(lines) // 10) * 10 :]))

def printBoard(board): # This function prints the board into the console
    printList = ["-" * (len(board) * 3)]
    printList += ["" for _ in range(len(board) * 3)]
    printList += ["-" * (len(board) * 3)]
    for i in range(len(board)):
        for j in range(len(board[i])):
            if board[i][j] != None:
                printSector = str(board[i][j]).split("\n")
            else:
                printSector = ["XXX", "XXX", "XXX"]
            for x in range(3):
                printList[i * 3 + x + 1] += printSector[x]
    printHugeString("\n".join(["|" + x + "|" for x in printList]))

BOARDSIZE = 11 # Stores what the width and height are for the board

def generatePaths(board, current, layerMax): # This function continually generates paths from a given input of
    starting positions that could be rooms
    layer = 0                                # This returns all the entrances that are open on each sides
    currentNorth = []
    currentSouth = []
    currentEast = []
    currentWest = []
```

```
while len(current) > 0: # Will continue to generate rooms until there are no open entrances left that are not
on the edges
    newCurrent = []
    for pos in current:
        north = False
        if board[pos[0]][pos[1]] != None: # Checks if there is a room already in that spot
            continue

        # Goes through each possible entrance and sees if it has to be open (Because a room next to it has the
 entrance open)
        # Or it randomises a possibility that it should be open
        pathCount = 0
        if pos[0] > 0 and board[pos[0] - 1][pos[1]] != None:
            if board[pos[0] - 1][pos[1]].enterences[1] == True:
                north = True
            pathCount += 1
        else:
            r = randint(0, 2)
            if r == 0:
                north = True
                pathCount += 1

        south = False
        if pos[0] < BOARDSIZE - 1 and board[pos[0] + 1][pos[1]] != None:
            if board[pos[0] + 1][pos[1]].enterences[0] == True:
                south = True
            pathCount += 1
        else:
            r = randint(0, 2)
            if r == 0:
                south = True
                pathCount += 1

        east = False
        if pos[1] < BOARDSIZE - 1 and board[pos[0]][pos[1] + 1] != None:
            if board[pos[0]][pos[1] + 1].enterences[3] == True:
                east = True
            pathCount += 1
        else:
            r = randint(0, 2)
            if r == 0:
                east = True
                pathCount += 1

        west = False
        if pos[1] > 0 and board[pos[0]][pos[1] - 1] != None:
            if board[pos[0]][pos[1] - 1].enterences[2] == True:
                west = True
            pathCount += 1
        else:
            r = randint(0, 2)
            if r == 0:
                west = True
                pathCount += 1

        # This checks to see if the room has enough entrances open to produce a big enough maze
        # If not it will randomise a few more entrances to be opened
        if pathCount == 1 and layer < layerMax:
            options = [north, south, east, west]
            r = randint(0, 2)
            c = 0
            for i in range(4):
                if options[i]:
                    continue
                if c == r:
                    options[i] = True
                    break
                c += 1
            r = randint(0, 2)
            if r != 2:
                r = randint(0, 2)
                c = 0
                for i in range(4):
                    if options[i]:
                        continue
                    if c == r:
                        options[i] = True
                        break
                    c += 1
            north = options[0]
```

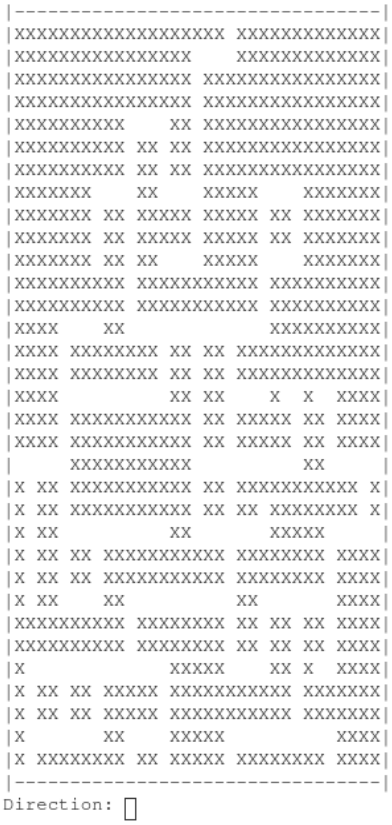
```
                south = options[1]
                east = options[2]
                west = options[3]

            # This does another check for rooms that are even closer to the centre, so there is less possibility
    of a maze that is extremely small being generated
            if pathCount == 2 and layer < layerMax - int(BOARDSIZE / 3):
                r = randint(0, 2)
                if r != 2:
                    options = [north, south, east, west]
                    r = randint(0, 2)
                    c = 0
                    for i in range(4):
                        if options[i]:
                            continue
                        if c == r:
                            options[i] = True
                            break
                        c += 1
                    north = options[0]
                    south = options[1]
                    east = options[2]
                    west = options[3]

            # This appends any new entrance made, without a room next to it to the next list of rooms to be
    generated
            if north and pos[0] > 0 and board[pos[0] - 1][pos[1]] == None:
                newCurrent.append((pos[0] - 1, pos[1]))

            if south and pos[0] < BOARDSIZE - 1 and board[pos[0] + 1][pos[1]] == None:
                newCurrent.append((pos[0] + 1, pos[1]))

            if east and pos[1] < BOARDSIZE - 1 and board[pos[0]][pos[1] + 1] == None:
                newCurrent.append((pos[0], pos[1] + 1))

            if west and pos[1] > 0 and board[pos[0]][pos[1] - 1] == None:
                newCurrent.append((pos[0], pos[1] - 1))

            # Creates the room and puts it onto the board
            board[pos[0]][pos[1]] = Sector(north, south, east, west)

        current = [x for x in newCurrent]

    # This finds all the entrances on the edge of the board
    for i in range(BOARDSIZE):
        if board[0][i] != None and board[0][i].enterences[0]:
            currentNorth.append(i)
        if board[-1][i] != None and board[-1][i].enterences[1]:
            currentSouth.append(i)
        if board[i][-1] != None and board[i][-1].enterences[2]:
            currentEast.append(i)
        if board[i][0] != None and board[i][0].enterences[3]:
            currentWest.append(i)

    return board, currentNorth, currentSouth, currentEast, currentWest


def generateBoard(): # This returns a board that has generated a maze, as well as all the entrances on the side
    board = [[None for _ in range(BOARDSIZE)] for _ in range(BOARDSIZE)]
    midPoint = BOARDSIZE // 2 + 1
    board[midPoint][midPoint] = Sector(True, True, True, True)
    current = [(midPoint - 1, midPoint), (midPoint, midPoint - 1), (midPoint + 1, midPoint), (midPoint, midPoint +
     1)]
    return generatePaths(board, current, int(BOARDSIZE * 2 / 3))

# These next functions are for moving the board, by deleting one row and adding another row and then calling the
    generation function
def moveNorth(board, current):
    newCurrent = [(0, x) for x in current]
    del board[-1]
    board.insert(0, [None for _ in range(BOARDSIZE)])
    return generatePaths(board, newCurrent, 5)

def moveSouth(board, current):
    newCurrent = [(BOARDSIZE - 1, x) for x in current]
    del board[0]
    board.append([None for _ in range(BOARDSIZE)])
    return generatePaths(board, newCurrent, 5)

def moveEast(board, current):
    newCurrent = [(x, BOARDSIZE - 1) for x in current]
    for i in range(BOARDSIZE):
        del board[i][0]
        board[i].append(None)

    return generatePaths(board, newCurrent, 2)

def moveWest(board, current):
    newCurrent = [(x, 0) for x in current]
    for i in range(BOARDSIZE):
        del board[i][-1]
        board[i].insert(0, None)

    return generatePaths(board, newCurrent, 5)

currentNorth = []
currentSouth = []
currentEast = []
currentWest = []

# This generates the board and allows the user to move in the different directions
board, currentNorth, currentSouth, currentEast, currentWest = generateBoard()
printBoard(board)
while True:
    inp = input("Direction: ").lower()
    if inp == "u":
        board, currentNorth, currentSouth, currentEast, currentWest = moveNorth(board, currentNorth)
    elif inp == "d":
        board, currentNorth, currentSouth, currentEast, currentWest = moveSouth(board, currentSouth)
    elif inp == "r":
        board, currentNorth, currentSouth, currentEast, currentWest = moveEast(board, currentEast)
    elif inp == "l":
        board, currentNorth, currentSouth, currentEast, currentWest = moveWest(board, currentWest)
    printBoard(board)
```

prototypes/MazeGen.py

2.2.3 Output

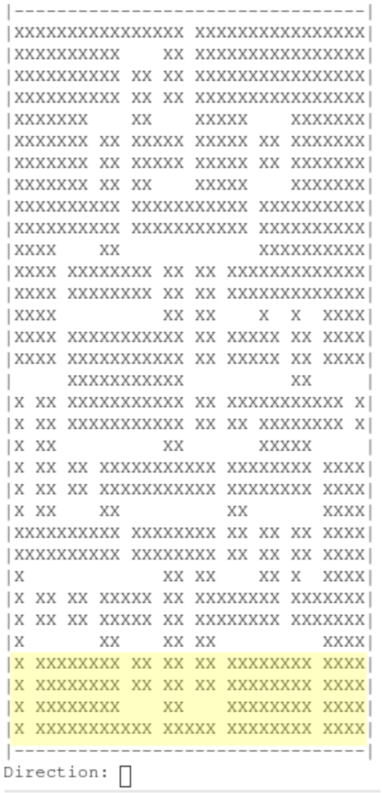
This is the output of the prototype, with the 'X' representing a wall and blank space represented as path the player can walk through. I have also shown the output of the maze when the player moves south and east. The new sections of the maze generated, after taking each step, is highlighted in yellow.

The first output



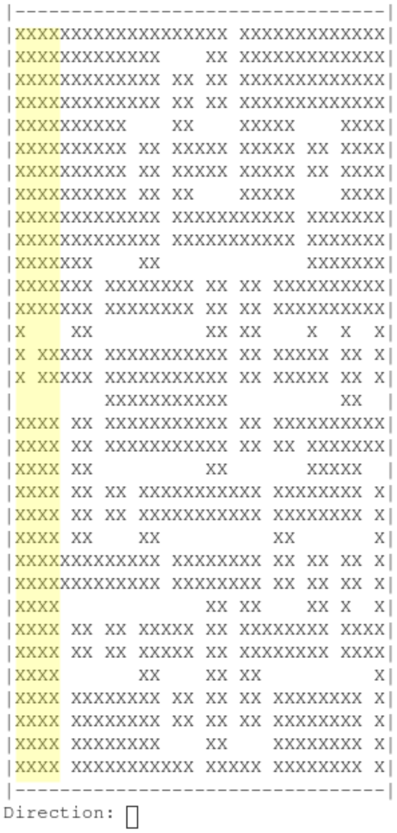
```
|---------------------------------|
|XXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXX|
|XXXXXXXXXXXXXXXX    XXXXXXXXXXXXX|
|XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX|
|XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX|
|XXXXXXXXXX    XX XXXXXXXXXXXXXXXX|
|XXXXXXXXXX XX XX XXXXXXXXXXXXXXXX|
|XXXXXXXXXX XX XX XXXXXXXXXXXXXXXX|
|XXXXXXX    XX    XXXXX    XXXXXXX|
|XXXXXXX XX XXXXX XXXXX XX XXXXXXX|
|XXXXXXX XX XXXXX XXXXX XX XXXXXXX|
|XXXXXXX XX XX    XXXXX    XXXXXXX|
|XXXXXXXXXX XXXXXXXXXXX XXXXXXXXXX|
|XXXXXXXXXX XXXXXXXXXXX XXXXXXXXXX|
|XXXX    XX             XXXXXXXXXX|
|XXXX XXXXXXXX XX XX XXXXXXXXXXXXX|
|XXXX XXXXXXXX XX XX XXXXXXXXXXXXX|
|XXXX          XX XX    X  X  XXXX|
|XXXX XXXXXXXXXXX XX XXXXX XX XXXX|
|XXXX XXXXXXXXXXX XX XXXXX XX XXXX|
|     XXXXXXXXXXX          XX     |
|X XX XXXXXXXXXXX XX XXXXXXXXXXX X|
|X XX XXXXXXXXXXX XX XX XXXXXXXX X|
|X XX          XX       XXXXX     |
|X XX XX XXXXXXXXXXX XXXXXXXX XXXX|
|X XX XX XXXXXXXXXXX XXXXXXXX XXXX|
|X XX    XX          XX       XXXX|
|XXXXXXXXXX XXXXXXXX XX XX XX XXXX|
|XXXXXXXXXX XXXXXXXX XX XX XX XXXX|
|X             XXXXX    XX X  XXXX|
|X XX XX XXXXX XXXXXXXXXXX XXXXXXX|
|X XX XX XXXXX XXXXXXXXXXX XXXXXXX|
|X       XX    XXXXX          XXXX|
|X XXXXXXXX XX XXXXX XXXXXXXX XXXX|
|---------------------------------|
Direction: 
```

After moving south (downwards)



```
|---------------------------------|
|XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX|
|XXXXXXXXXX    XX XXXXXXXXXXXXXXXX|
|XXXXXXXXXX XX XX XXXXXXXXXXXXXXXX|
|XXXXXXXXXX XX XX XXXXXXXXXXXXXXXX|
|XXXXXXX    XX    XXXXX    XXXXXXX|
|XXXXXXX XX XXXXX XXXXX XX XXXXXXX|
|XXXXXXX XX XXXXX XXXXX XX XXXXXXX|
|XXXXXXX XX XX    XXXXX    XXXXXXX|
|XXXXXXXXXX XXXXXXXXXXX XXXXXXXXXX|
|XXXXXXXXXX XXXXXXXXXXX XXXXXXXXXX|
|XXXX    XX             XXXXXXXXXX|
|XXXX XXXXXXXX XX XX XXXXXXXXXXXXX|
|XXXX XXXXXXXX XX XX XXXXXXXXXXXXX|
|XXXX          XX XX    X  X  XXXX|
|XXXX XXXXXXXXXXX XX XXXXX XX XXXX|
|XXXX XXXXXXXXXXX XX XXXXX XX XXXX|
|     XXXXXXXXXXX          XX     |
|X XX XXXXXXXXXXX XX XXXXXXXXXXX X|
|X XX XXXXXXXXXXX XX XX XXXXXXXX X|
|X XX          XX       XXXXX     |
|X XX XX XXXXXXXXXXX XXXXXXXX XXXX|
|X XX XX XXXXXXXXXXX XXXXXXXX XXXX|
|X XX    XX          XX       XXXX|
|XXXXXXXXXX XXXXXXXX XX XX XX XXXX|
|XXXXXXXXXX XXXXXXXX XX XX XX XXXX|
|X             XX XX    XX X  XXXX|
|X XX XX XXXXX XX XXXXXXXX XXXXXXX|
|X XX XX XXXXX XX XXXXXXXX XXXXXXX|
|X       XX    XX XX          XXXX|
|X XXXXXXXX XX XX XX XXXXXXXX XXXX|
|X XXXXXXXX XX XX XX XXXXXXXX XXXX|
|X XXXXXXXX    XX    XXXXXXXX XXXX|
|X XXXXXXXXXXX XXXXX XXXXXXXX XXXX|
|---------------------------------|
Direction: 
```

After moving east (leftwards)



```
|---------------------------------|
|XXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXX|
|XXXXXXXXXXXXX    XX XXXXXXXXXXXXX|
|XXXXXXXXXXXXX XX XX XXXXXXXXXXXXX|
|XXXXXXXXXXXXX XX XX XXXXXXXXXXXXX|
|XXXXXXXXXX    XX    XXXXX    XXXX|
|XXXXXXXXXX XX XXXXX XXXXX XX XXXX|
|XXXXXXXXXX XX XXXXX XXXXX XX XXXX|
|XXXXXXXXXX XX XX    XXXXX    XXXX|
|XXXXXXXXXXXXX XXXXXXXXXXX XXXXXXX|
|XXXXXXXXXXXXX XXXXXXXXXXX XXXXXXX|
|XXXXXXX    XX             XXXXXXX|
|XXXXXXX XXXXXXXX XX XX XXXXXXXXXX|
|XXXXXXX XXXXXXXX XX XX XXXXXXXXXX|
|X    XX          XX XX    X  X  X|
|X XXXXX XXXXXXXXXXX XX XXXXX XX X|
|X XXXXX XXXXXXXXXXX XX XXXXX XX X|
|        XXXXXXXXXXX          XX  |
|XXXX XX XXXXXXXXXXX XX XXXXXXXXXX|
|XXXX XX XXXXXXXXXXX XX XX XXXXXXX|
|XXXX XX          XX       XXXXX  |
|XXXX XX XX XXXXXXXXXXX XXXXXXXX X|
|XXXX XX XX XXXXXXXXXXX XXXXXXXX X|
|XXXX XX    XX          XX       X|
|XXXXXXXXXXXXX XXXXXXXX XX XX XX X|
|XXXXXXXXXXXXX XXXXXXXX XX XX XX X|
|XXXX             XX XX    XX X  X|
|XXXX XX XX XXXXX XX XXXXXXXX XXXX|
|XXXX XX XX XXXXX XX XXXXXXXX XXXX|
|XXXX       XX    XX XX          X|
|XXXX XXXXXXXX XX XX XX XXXXXXXX X|
|XXXX XXXXXXXX XX XX XX XXXXXXXX X|
|XXXX XXXXXXXX    XX    XXXXXXXX X|
|XXXX XXXXXXXXXXX XXXXX XXXXXXXX X|
|---------------------------------|
Direction: 
```

2.3 A* Algorithm

2.3.1 Explanation

This is a common algorithm used for finding the shortest route between two points because speed while also being very versatile.

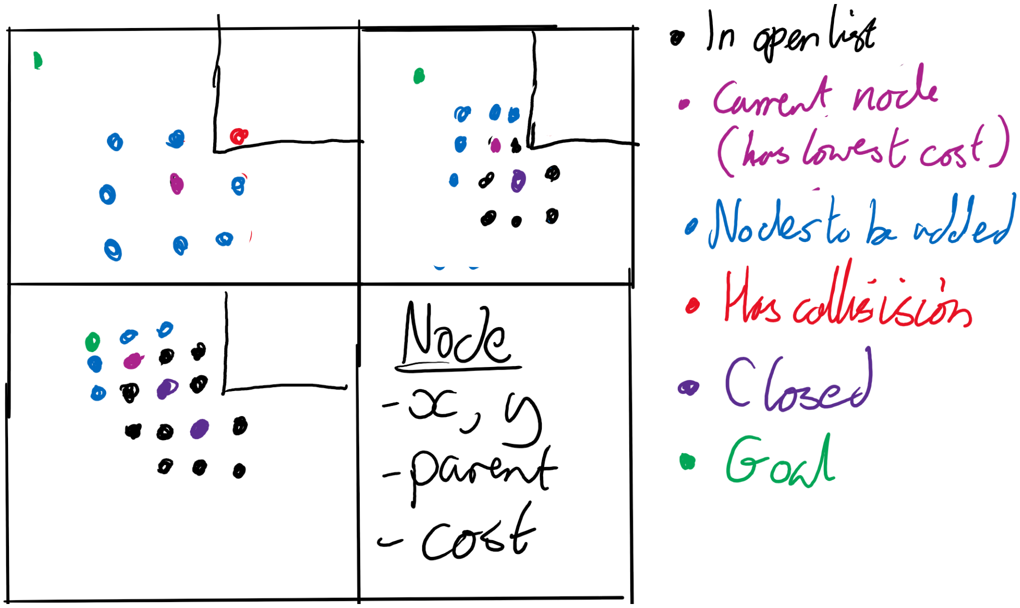


Figure 9: Drawing describing the process of how the A* algorithm finds the shortest route

The final box shows what the Node class needs to store, in order for this to work, with the different colours representing different states a node can be in, labelled on the side.

2.3.2 Prototype

2.4 Graphical Design

2.4.1 Overall Design

The overall design is (as shown below) to have a simple GUI system where the player can see what current weapons they have available to them as well as their health, then a simple button at the top to pause the game and go back to a menu screen. Furthermore, the camera will be facing downwards and is above the player. This will make it easier to create a rendering system and allows the player to explore in any direction (except up and down). Items will be able to be found on the ground, with rendered with their texture, and not a back or anything to allow anyone to easily discern between the different objects.

Each room will be simple, with different objects in the center, for example this room has a chest in the center which the player will be able to interact with and grab items out of. Then there can be entrances at the top, bottom, left and right of the room and will be generated using the method talked above (in the maze generation section).

Please note that the weapons, potions, and books were gained from Shikashi's Fantasy Icons Pack v2 by Matt Firth. Also the tiles for the rooms were altered slightly to fit the needed tiles however the textures were originally from RPG Nature Tileset - Seasons by Stealthix. However, I will be making the design for each of the npcs as I cannot find any that will fit the design.

2.4.2 NPCs

2.4.3 Weapons

2.4.4 Potions

2.4.5 Food and Books

2.4.6 Tiles

2.5 General Design

2.5.1 Stats

Each stat will influence part of how you play the game.

- Strength - Directly effects the damage an entity can do.
- Agility - Increases speed of himself and followers and decreases the speed of attacks.
- Health - Directly effects how long it takes for you to die.

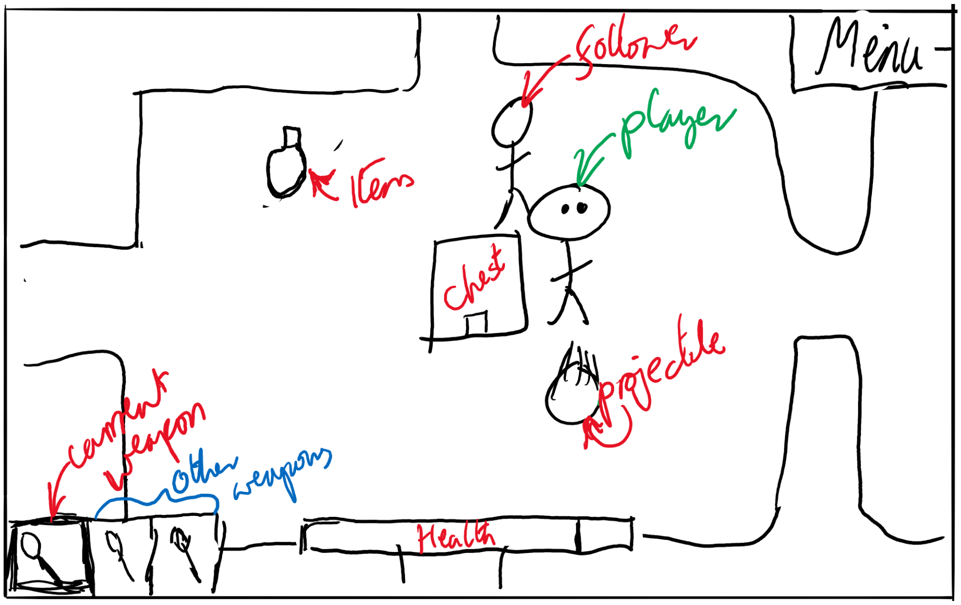


Figure 10: Overall Design plan

- Combat Ability - Influences the likelihood of higher damages when attacking
- Stamina - Influences the accuracy and damage when attacking and directly influences the amount a mob can carry.
- Boredom - Decreases speed and accuracy. This is decreased through finding items and reading books. This is also contagious between a mob's followers.
- Minimum attack damage - This is damage done when a mob has no weapon.
- Attractiveness - Influences the maximum number of followers each mob can have, however if a mob is following another, this is set to 0.

2.5.2 Rooms

Each room has their own effects and contains different objects. This will create more variety when exploring.

- Trap Room - This will contain a trap, which can harm or kill the player or a follower. However, there should also be a chance for the player to avoid the trap through pressing some keys at the right time.
- Treasure Room - This will contain a chest, containing items which the player can collect and distribute to followers.
- Stair Room - This will contain stairs that lead to the next level.
- Trapdoor Room - This can be disguised as a trap room that would cause the player to fall down to the next level.
- Hidden treasure room, this is a room that all the entrances are hidden until the player actively reveals the entrance.
- Enemy room - this should contain an enemy inside the room, which will start attacking when the player walks in. Also the entrances should be closed (this could be the rooms around it entrances closing to create to make sure the player does not get stuck)

2.6 Structure Overview

2.6.1 Singletons

In the program, there will some key classes that everything will need to have access to. So, to combat this, those classes will be singletons (classes with only one instance ever created). Then these classes will have a get function which will return that single instance and so anything can call it and have access to the functions it needs to. To make this easier, I will also create static versions of each function, which act as a reference to the Implemented function by calling the get function. This will make the code look a lot more readable and thus easier to debug.

The classes that will be singletons will be:

- Application - This will control all the layers and store the key information needed for creating a window.
- Render - This will control all the rendering
- Random - This will be the random generator for all the numbers, as in C++ you should only generate a generator once.

- Log - This will be for logging everything to a file and outputting it to the terminal in debug mode
- ShaderEffectsManager - This will control any shader effects that are applied on any layer in the application, storing the effects and handling IDs

2.6.2 Layers

For rendering and updating, I will be using a layering system, where each layer will have its own effects and control different parts of the game, for example the actual level layer and the GUI layer. This will allow more control over what receives events and what order they get them in, as well has the order in which things are rendered.

As mentioned above, these layers will be stored by the application function. The application will be in charge of the flow of information and knowing which layers are overlays and which are not.

2.6.3 Rendering System

For the rendering system, each class that needs to be rendered will have a render function which will be called every frame. This will then call the relevant function in the render class to render itself. These functions should then send the information into a buffer, which then will only be rendered once the appropriate render function is called. This should be automatically called by the Application after each layer. This function will then convert all the information stored on the buffers into vertices which then will be rendered using the correct shader to get the intended effect (This might mean that I have to have multiple buffers for different objects e.g. text and a coloured rectangle)

2.6.4 Flow

The control of the frame rate and the updates per second will be controlled by a standalone function in the main file. This will make sure the ups (updates per second) will be a continuous 60 ups, while the fps (frames per second) will run as many times per second as possible. This function will call the relevant update and render function in the application class, which will then call the function on every layer. This should mean that everything in any layer is updated and rendered at the correct times.

2.7 Classes

2.7.1 Application

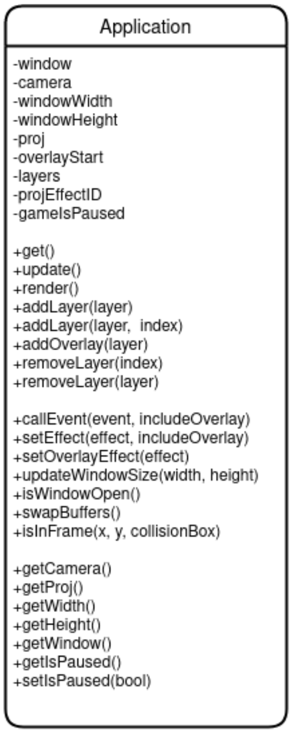


Figure 11: Application class (singleton)

Variables

Variable Name	**Description**
window	Stores the application window itself
camera	Stores the camera that effects the non-overlay layers
windowWidth	Stores the width of the window
windowHeight	Stores the height of the window
proj	Stores the projection map matrix for the window
overlayStart	Stores the position at which the overlay layers start in the layer stack
layers	Stores the layer stack
projEffectID	Stores the ID of the projection effect for all the layers
gameIsPaused	Stores whether the game is paused or not

Functions

Function Name	**Parameters**	**Description**
get		Returns the single instance of the Application
update		Calls the update function on all the layers
render		Calls the render function on all the layers
addLayer	Layer you wish to add	Adds a layer to the stack (under the overlays)
addOverlay	Overlay (as a layer) you wish to add	Adds a layer to the stack on top of all the other layers
removeLayer	Either the index of the layer or the layer you wish to remove	Removes a layer from the stack
callEvent	Event and boolean to tell it whether to include the overlay	Calls the event on every layer until one of them uses it
setEffect	Effect and boolean to tell it whether to include the overlay	Sends the effect to every layer until one of them uses it
setOverlay	An effect to send	Sends an effect through only the overlays until one of them uses it
updateWindowSize	The new width and height of the window	Updates the window size stored in the Application
isWindowOpen		Returns whether the application is open or not
swapBuffers		Swaps the buffers of the application (for rendering)
isInFrame	x and y of the object and the collision box of the object	Acts as a go between for the camera's isInFrame function
getCamera		Returns the camera used for the non-overlay layers
getProj		Returns the projection map for the application window
getWidth		Returns the width of the application window
getHeight		Returns the height of the application window
getWindow		Returns the openGL window
getIsPaused		Returns whether the game is paused or not (preventing the non-overlay functions from being updated)
setIsPaused	boolean that the isPaused variable is set to	Sets the isPaused variable

2.7.2 Render

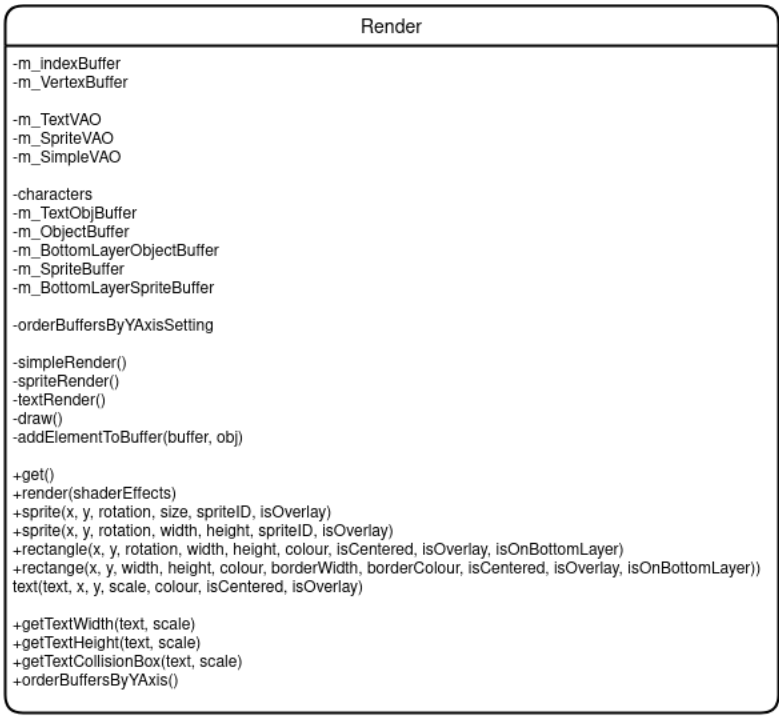

Render
-m_indexBuffer
-m_VertexBuffer
-m_TextVAO
-m_SpriteVAO
-m_SimpleVAO
-characters
-m_TextObjBuffer
-m_ObjectBuffer
-m_BottomLayerObjectBuffer
-m_SpriteBuffer
-m_BottomLayerSpriteBuffer
-orderBuffersByYAxisSetting
-simpleRender()
-spriteRender()
-textRender()
-draw()
-addElementToBuffer(buffer, obj)
+get()
+render(shaderEffects)
+sprite(x, y, rotation, size, spriteID, isOverlay)
+sprite(x, y, rotation, width, height, spriteID, isOverlay)
+rectangle(x, y, rotation, width, height, colour, isCentered, isOverlay, isOnBottomLayer)
+rectange(x, y, width, height, colour, borderWidth, borderColour, isCentered, isOverlay, isOnBottomLayer))
text(text, x, y, scale, colour, isCentered, isOverlay)
+getTextWidth(text, scale)
+getTextHeight(text, scale)
+getTextCollisionBox(text, scale)
+orderBuffersByYAxis()

Figure 12: Render class (singleton)

Variables

Variable Name	Description
m_IndexBuffer	Stores the index buffer used for drawing all the vertices in the right order
m_VertexBuffer	Stores the buffer used for all the rendering
m_TextVAO	Stores the vertex array for drawing text
m_SpriteVAO	Stores the vertex array for drawing sprites
m_SimpleVAO	Stores the vertex array for drawing rectangles
characters	Stores all the characters textures and information needed to draw each character of a text font
m_TextObjBuffer	Stores all the objects that need to be rendered in this frame
m_ObjectBuffer	Stores all the objects that need to be rendered in this frame
m_BottomLayerObjectBuffer	Stores all the objects that need to be rendered before anything else on this frame
m_SpriteBuffer	Stores all the sprite objects that need to be rendered this frame
m_BottomLayerSpriteBuffer	Stores all the sprite objects that need to be rendered before anything else on this frame
orderBuffersByYAxisSetting	Boolean that tells whether the buffers should be sorted by the Y position of objects
m_SpriteShader	Stores the shader for rendering sprites
m_TextShader	Stores the shader for rendering text
m_SimpleShader	Stores the shader for rendering coloured rectangles

Functions

Function Name	**Parameters**	**Description**
simpleRender		Renders everything stored in m_ObjectBuffer and m_BottomLayerObjectBuffer
spriteRender		Renders everything stored in sprite buffers
textRender		Renders everything in m_TextObjBuffer
draw	Takes in a VAO to render with	Draws all the vertices stored in m_VertexBuffer onto the screen with a given VAO
addElementToBuffer	Takes in the buffer to add the object to and the object	Adds an object onto a buffer taking into account orderBuffersByYAxisSetting
get		Returns the only instance of Render
render	Takes in a list of shaderEffects to apply to the shaders	Sets the shaderEffects to the shaders and then calls the other render functions
sprite	Takes in all the information needed for rendering (Can use a size or a specific value for the width and the height)	Adds TexturedObject object to the sprite buffers
rectangle	Takes in all the information to render a rectangle, a second function is made for rendering rectangles with a border	Adds ColouredObject to the object buffers
text	Takes all the information in to render text on the screen	Adds TextObject to the text buffer
getTextWidth	Takes in the text and the scale	Returns the width of a given text at a given scale
getTextHeight	Takes in the text and the scale	Returns the height of the text at a given scale
getTextCollisionBox	Takes in the text and the scale	Returns the collision box of the text at a given scale
orderBuffersByYAxis		Turns the setting on

2.7.3 Other Singletons

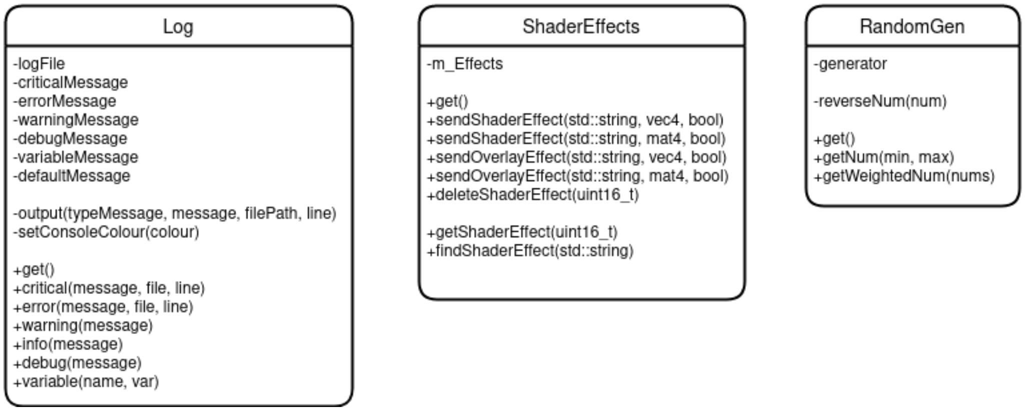

Log
-logFile
-criticalMessage
-errorMessage
-warningMessage
-debugMessage
-variableMessage
-defaultMessage
-output(typeMessage, message, filePath, line)
-setConsoleColour(colour)
+get()
+critical(message, file, line)
+error(message, file, line)
+warning(message)
+info(message)
+debug(message)
+variable(name, var)
ShaderEffects
-m_Effects
+get()
+sendShaderEffect(std::string, vec4, bool)
+sendShaderEffect(std::string, mat4, bool)
+sendOverlayEffect(std::string, vec4, bool)
+sendOverlayEffect(std::string, mat4, bool)
+deleteShaderEffect(uint16_t)
+getShaderEffect(uint16_t)
+findShaderEffect(std::string)
RandomGen
-generator
-reverseNum(num)
+get()
+getNum(min, max)
+getWeightedNum(nums)

Figure 13: Other singleton classes

Log

Variables

Variable Name	Description
logFile	Stores the filename and location of the log file
criticalMessage	Stores the identifier for a critical message
errorMessage	Stores the identifier for an error message
warningMessage	Stores the identifier for a warning message
debugMessage	Stores the identifier for a debug message
variableMessage	Stores the identifier for a message with a variable
defaultMessage	Stores the default identifier

Functions

Function Name	Parameters	Description
output	Takes in the identifier and the message as well as the filepath and the line of where the log occurred	Outputs the message in the correct format
setConsoleColour	A colour	Sets the console to the colour given (in debug mode for the terminal)
get		Returns the only instance of the Log class
critical	Takes in the message and information for debugging	Uses the output function to output a critical message
error	Takes in the message and information for debugging	Uses the output function to output an error message
warning	Takes in the message	Uses the output function to output a warning
info	Takes in a message	Uses the output function to output a message
debug	Takes in a message	Uses the output function to output a debug message
variable	Takes in the name of the variable and the variable	Uses the output function to output a variable

ShaderEffectsManager

Variables

Variable Name	Description
m_Effects	Stores all the effects that are currently in use in the application

Functions

Function Name	**Parameters**	**Description**
get		Returns the only instance of the ShaderEffectsManager class
sendShaderEffect	The name of the effect and the effect (in vector or matrix form) and boolean to say whether it should include the overlays	Creates and sends the effects through the layers
sendOverlayEffect	The name of the effect and the effect (in vector or matrix form)	Creates and sends an effect through only the overlay layers
deleteShaderEffect	The ID of the effect	Deletes the effect and sends a message to all the layers to inform them that effect has been deleted
getShaderEffect	the ID of the effect	Returns the effect associated with that ID
findShaderEffect	the name of the effect	Finds and returns the ID of the effect with that variable name

RandomGen

Variables

Variable Name	**Description**
generator	Stores the generator used for all the random number generating (as described in the C++ documentation this should only be created once for each program)

Functions

Function Name	**Parameters**	**Description**
reverseNum	a number	Returns the number in reverse, used for generating the generator
get		Returns the only instance of the RandomGen class
getNum	Range for the random number	Returns a random number within the range
getWeightedNum	list of probabilities (should all add up to one)	Returns a random index of the list

2.7.4 Layers

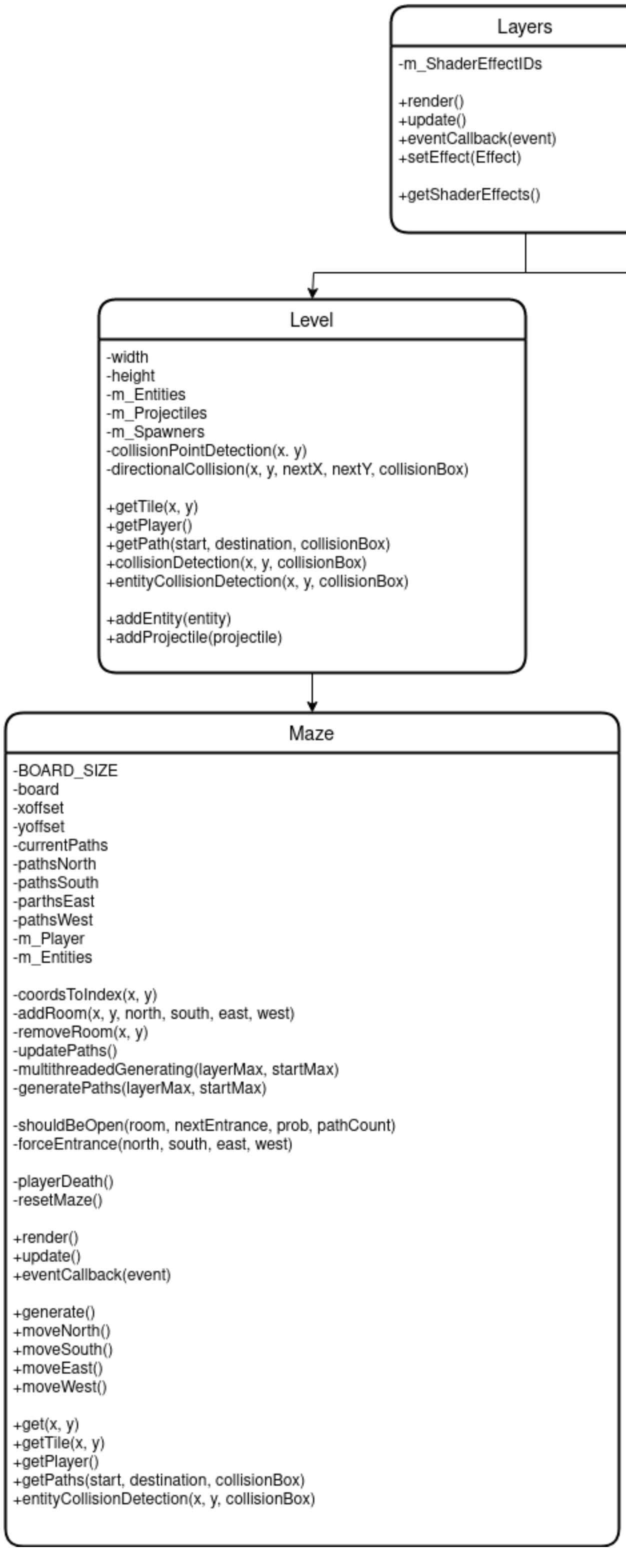


Figure 14: Layer subclasses

Layers

Variables

Variable Name	**Description**
m_ShaderEffectIDs	Stores the effects the layer needs when rendering

Functions

Function Name	**Parameters**	**Description**
render		Renders the layer
update		Updates the layer
eventCallback	event that has happened	Allows the layer to interact with events
setEffect	effect	Sets an effect onto the layer (Will probably be an effect for the shader)
getShaderEffects		Returns the shader effects for the layer

GUILayer

Variables

Variable Name	**Description**
m_ConnectedLevel	Stores the level it is connected to
m_Objects	Stores the objects that are involved in the menu

Functions

Function Name	**Parameters**	**Description**
addMenuObject	MenuObject to add	Adds a given menu object to the list of objects
setConnectedLevel	Level to connect to	Connects the layer to a given level (this does not need to be set - only for menus interacting with the game)
getObjects		Returns the list of objects that are involved in the menu
getConnectedLevel		Returns the level the layer is connected to

Level

Variables

Variable Name	**Description**
m_Player	Stores the player on that level
width	Stores the width of the level (in terms of tiles)
height	Stores the height of the level (in terms of tiles)
m_Entities	Stores a list of all the entities in the level
m_Projectiles	Stores all the projectiles in the level
m_Spawners	Stores all the current spawners in the level

Functions

Function Name	**Parameters**	**Description**
collisionPointDetection	x and y of a point	Calculates whether a point is within a solid tile
directionalCollision	current x and y and the next x and y and the collision box of the object	Calculates whether an object is going to collide with any tile within the level
getTile	Tile x and y position in the level	Returns the tile at that point in time
getPlayer		Returns the player
getPath	start position, end position and the collisionBox of the object	Returns a path of the shortest route between two points (using A* algorithm)
collisionDetection	x, y and collisionBox of an object	returns whether it has collided with anything
entityCollisionDetection	x, y and collisionBox of an object	returns whether it has collided with an entity in the level
addEntity	entity	Adds an entity to the level
addProjectile	projectile	Adds a projectile to the level
addSpawner	spawner	Adds a spawner to the level

Maze

Variables

Variable Name	Description
BOARD_SIZE	Static, constant variable that stores the width of the maze (in rooms)
board	Stores the list of rooms in the maze
xoffset	Stores the offset in the x direction for the top left corner of the maze
yoffset	Stores the offset in the y direction for the top left corner of the maze
currentPaths	Stores the current available paths the maze can generate in (used for generating the maze)
pathsNorth	Stores the paths that can be generated when the maze moves north
pathsSouth	Stores the paths that can be generated when the maze moves south
pathsEast	Stores the paths that can be generated when the maze moves east
pathsWest	Stores the paths that can be generated when the maze moves west

Functions

Function Name	Parameters	Description
coordsToIndex	x and y position of a room	Converts a 2d coordinates for a room into an index of where it is in the board variable
addRoom	position and booleans for each entrance it could have	This adds a room at given coordinates, randomising what room it is and adding entities into it
removeRoom	x and y position of the room	This removes a room from the maze
updatePaths		This updates the paths variables by resetting them and looking for new ones
multithreadedGenerating	The maximum layers for a boosted effect of generating and start maximum probability	This sets up everything needed to have the generating of the maze in another thread
generatePaths	The maximum layers for a boosted effect of generating and start maximum probability	This is the function written in the prototype transferred for generating the maze using the currentPaths variable
shouldBeOpen	room, the next entrance, the probability of the entrance and count of how many entrances are already open	This returns an entrance state and chooses whether the entrance, should be open or closed (or it is closed but it could be opened)
forceEntrance	reference to the boolean values that store whether the entrance is going to be open or not	This will force an entrance, when the program believes there needs to be another entrance when generating
playerDeath		This is the function that handles everything when the player dies
resetMaze		This deals with resetting the whole maze
generate		This is the function to call to generate a new maze
moveNorth		This handles the maze moving to the north (and generates new rooms)
moveSouth		This handles the maze moving to the south (and generates new rooms)
moveEast		This handles the maze moving to the east (and generates new rooms)
moveWest		This handles the maze moving to the west (and generates new rooms)
get	x and y pos of a room	This returns a room at the given coordinates

2.7.5 Entities

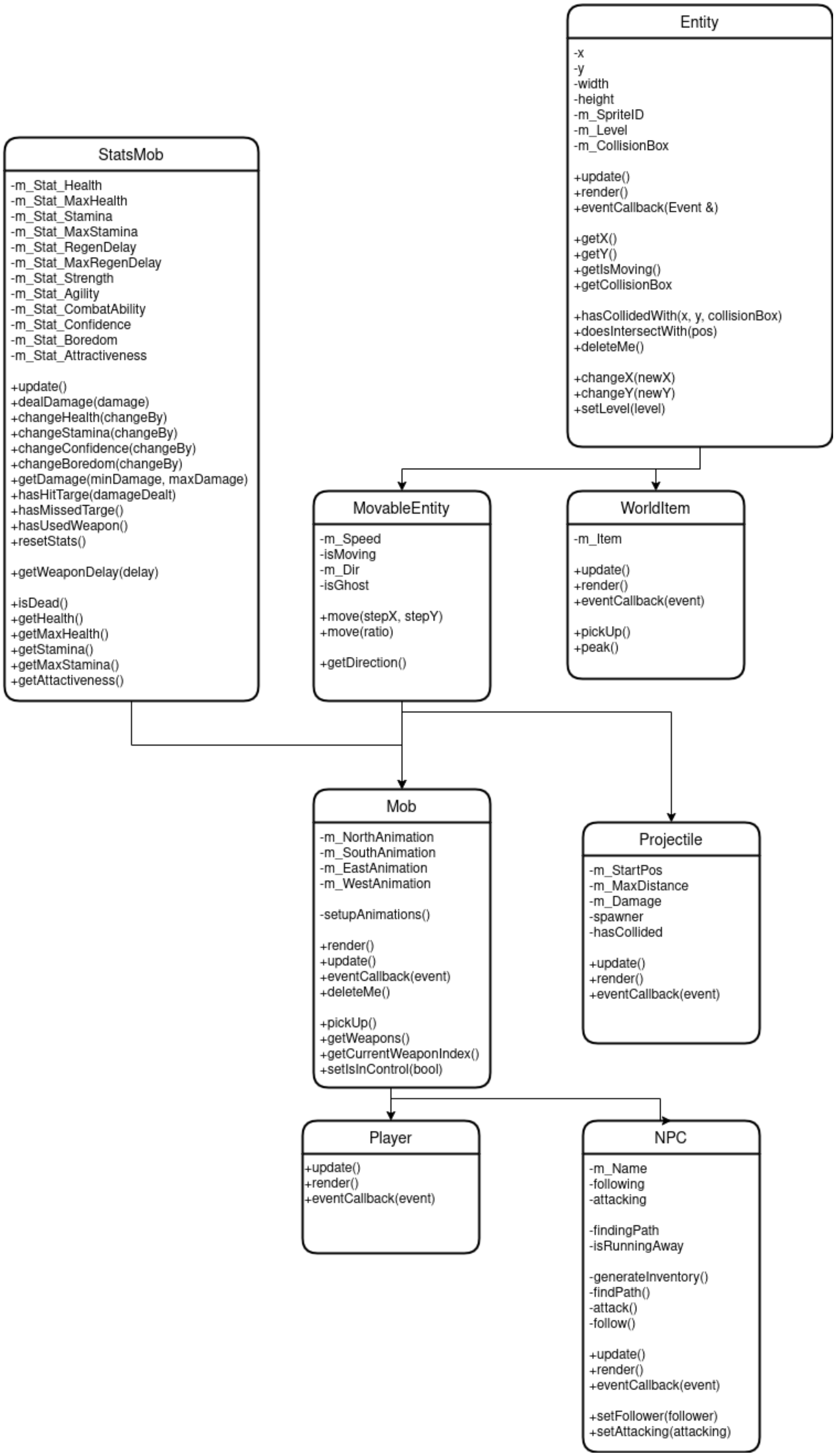


Figure 15: Entity subclasses and StatsMob

Entity

Variables

Variable Name	Description
x	Stores the x position of the entity
y	Stores the y position of the entity
width	Stores the width of the entity
height	Stores the height of the entity
m_SpriteID	Stores the sprite ID of the entity
m_Level	Stores the level the entity is located in
m_CollisionBox	Stores the collision box of the entity

Functions

Function Name	Parameters	Description
update		Updates the entity
render		Renders the entity
eventCallback	Event	This allows entities to listen for events
getX		Returns x position
getY		Returns y position
getIsMoving		Returns whether the entity is moving
getCollisionBox		Returns the collision box
hasCollidedWith	position and collisionBox of an object	returns whether its collision box intersects with theirs
doesIntersectWith	position of point	returns whether or not that point is inside its collision box
deleteMe		Returns whether the entity should be deleted
changeX	new x position	changes the x position of the entity
changeY	new y position	changes the y position of the entity
setLevel	level the entity is in	changes the level the entity is currently in

WorldItem

Variables

Variable Name	Description
m_Item	Stores the item that it is carrying

Functions

Function Name	Parameters	Description
pickUp		returns the item and removes it from its storage
peak		returns the item however doesn't remove it from its storage

MovableEntity

Variables

Variable Name	Description
m_Speed	Stores the maximum speed it can travel at
isMoving	Stores whether it is currently moving or not
m_Dir	Stores the direction it is travelling in
isGhost	Stores whether it ignores collision

Functions

Function Name	Parameters	Description
move	Can either be a change in x and y or a ratio (using its maximum speed)	This moves the object, checking for collisions and taking into account its maximum speed
getDirection		Returns the current direction of the entity

Projectile

Variables

Variable Name	Description
m_StartPos	Stores the start position of the projectile
m_MaxDistance	Stores the maximum distance the projectile can travel before being deleted
m_Damage	Stores the maximum damage the projectile can do
spawner	Stores the Mob who spawned the projectile
hasCollided	Stores whether it has collided with anything

StatsMob

Variables

Variable Name	Description
m_Stat_Health	Stores the health of the mob
m_Stat_MaxHealth	Stores the max health of the mob
m_Stat_Stamina	Stores the stamina of the mob
m_Stat_MaxStamina	Stores the max stamina of the mob
m_Stat_RegenDelay	Acts as a countdown to when the mob can start to regen its stats
m_Stat_MaxRegenDelay	Stores the maximum value of the regen delay counter
m_Stat_Strength	Stores the strength of mob
m_Stat_Agility	Stores the agility of the mob
m_Stat_CombatAbility	Stores the combat ability of the mob
m_Stat_Confidence	Stores the confidence of the mob (out of 100)
m_Stat_Boredom	Stores the boredom of the mob (out of 100)
m_Stat_Attractiveness	Stores the amount of followers the mob can have

Functions

Function Name	Parameters	Description
update		Updates the regen delay and handles regeneration of the mob's stats
dealDamage	max damage of weapon	Deals damage to the mob, taking into account their stats
changeHealth	changeBy	Changes the health (ignoring stats)
changeStamina	changeBy	Changes the stamina
changeConfidence	changeBy	Changes the confidence
changeBoredom	changeBy	Changes the boredom
getDamage	min and max damage of a weapon	Returns the damage the weapon should do taking into account the mob's stats
hasHitTarget	damage dealt	Increases the stats based on how much damage a weapon did
hasMissedTarget		Changes stats for missing the targe
hasUsedWeapon		Resets the regen delay (cannot regen while attacking)
resetStats		Resets stats
getWeaponDelay	max delay of weapon	Returns the delay on a weapon taking into account its stats
isDead		Returns true if the health is 0
getHealth		Returns the Mob's health
getMaxHealth		Returns the max health of the mob
getStamina		Returns the stamina of the mob
getMaxStamina		Returns the max stamina of the mob
getAttractiveness		Returns the attractiveness of a mob

Mob

Variables

Variable Name	Description
m_NorthAnimation	Stores the animation sprite for walking north
m_SouthAnimation	Stores the animation sprite for walking south
m_EastAnimation	Stores the animation sprite for walking east
m_WestAnimation	Stores the animation sprite for walking west
m_Weapons	Stores all the weapons of the mob
currentWeapon	Stores the current active weapon
m_Inventory	Stores the inventory of the mob
isInControl	States whether the mob is inControl of its actions

Functions

Function Name	Parameters	Description
setupAnimations		Initialises the animations for each direction
pickUp	Item to pick up	Adds an item into the inventory
getWeapons		Returns the weapons
getCurrentWeaponIndex		Returns the current weapon index
getInventory		Returns the inventory
setIsInControl	bool	Sets isInControl

Player

The player, only overrides classes its parent classes to achieve functionality

NPC

Variables

Variable Name	**Description**
m_Name	Stores the name of the follower/enemy
following	Stores the entity that it is following
attacking	Stores the entity that it is attacking
findingPath	Stores whether it is currently finding a path to take
isRunningAway	Stores whether it is running away from its enemy

Functions

Function Name	**Parameters**	**Description**
generateInventory		Generates the inventory of the NPC
findPaths		Finds the quickest route to the entity it is following
attack		Runs algorithm for attacking
follow		Runs algorithm for following
setFollower	follower	Sets the entity it is following
setAttacking	attacking	Sets the entity it is attacking

2.7.6 Maze objects

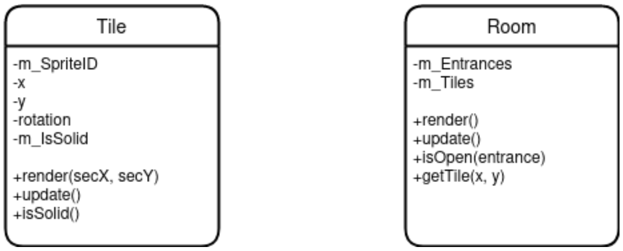


Figure 16: Classes for the design of the maze

Tile

Variables

Variable Name	Description
m_SpriteID	Stores the sprite ID of the tile
x	Stores the x axis relative to the room it is located in
y	Stores the y axis relative to the room it is located in
rotation	Stores the rotation of the tile
m_IsSolid	Stores whether it is solid or not

Functions

Function Name	Parameters	Description
render	The x and y coordinates of the room it is in	Renders the tile
update		Updates the tile (This is not really used as I do not have any animated tiles)
isSolid		Returns m_IsSolid

Room

Variables

Variable Name	Description
m_Entrances	Stores the entrances and whether they are open or not
m_Tiles	Stores all the tiles as a grid

Functions

Function Name	Parameters	Description
render		Renders all the tiles
update		Updates the room
isOpen	Entrance (this is its own type)	returns whether an entrance is open
getTile	x and y position	Returns a tile at the give coordinates

2.7.7 Rendering Utils

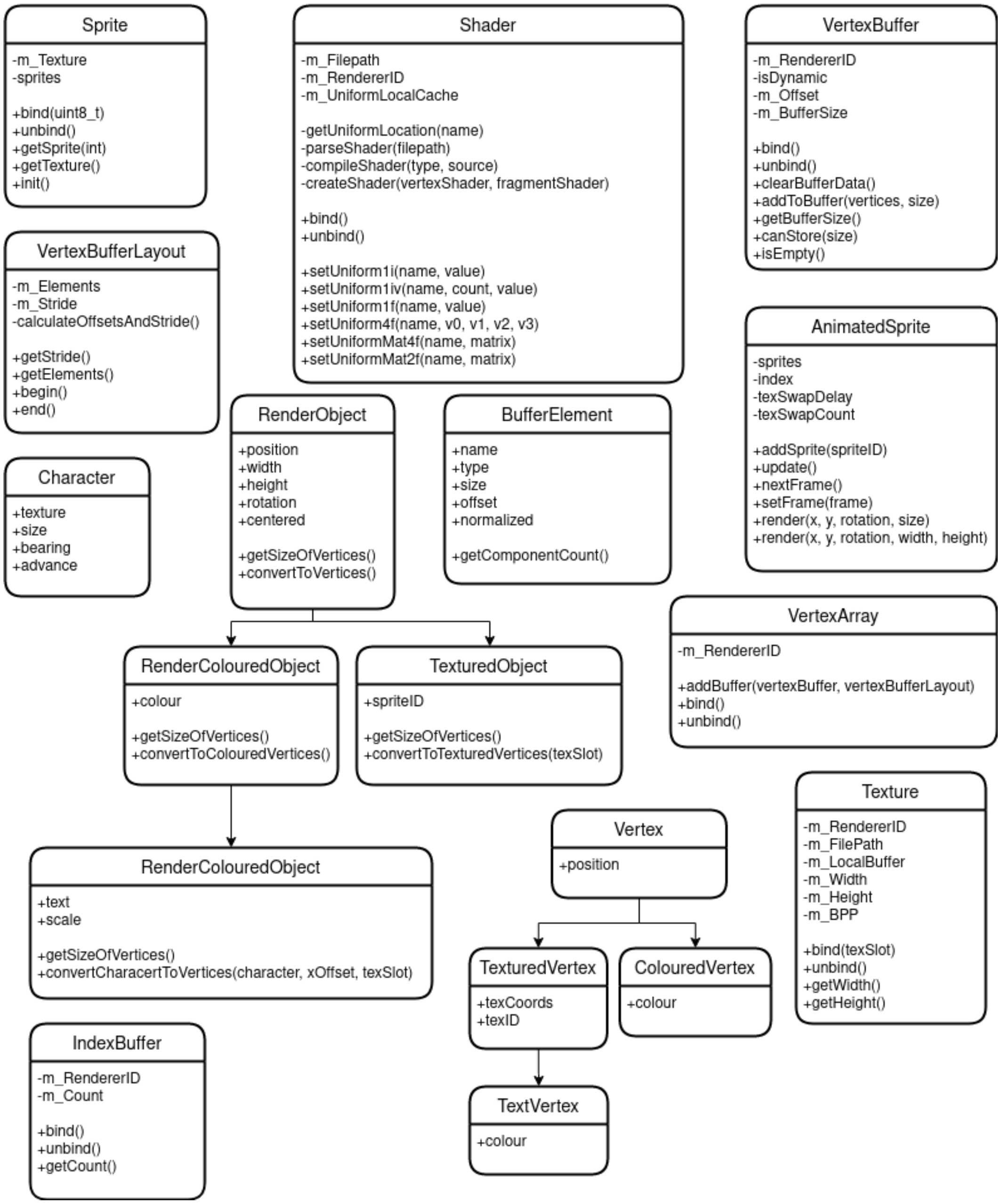


Figure 17: Classes involved in rendering

Sprite

Variables

Variable Name	Description
m_Texture	Stores the texture for that sprite
sprites	Array that stores all the current sprites in the program

Functions

Function Name	Parameters	Description
bind	Slot ID	Binds the texture at a given slot
unbind		Unbinds a given its texture
getSprite	Sprite ID	returns a sprite in the list
getTexture		Returns its texture
init		Run at the start of the program to initialise all the sprites

Shader

Variables

Variable Name	**Description**
m_Filepath	Used for debugging - Stores the filepath of the shader
m_RenderID	openGL ID used for interactions with the shader
m_UniformLocalCache	Stores all the locations of the uniforms in the shader for quick access

Functions

Function Name	**Parameters**	**Description**
getUniformLocation	name of the variable	returns the location of the variable in the shader
parseShader	filepath to the shader	returns vertex and buffer shader from the filepath
compileShader	type of shader and the source of the shader	compiles the shader and returns ID
createShader	vertex and fragment shader code	compiles and links the shader
bind		Binds the shader
unbind		Unbinds the shader
setUniform1i	name of the variable and the integer	Attaches the value to the uniform
setUniform1iv	name of the variable and how many is in the list, then pointer to first element	Attaches the value to the uniform
setUniform1f	name of the variable then the float	Attaches the value to the uniform
setUniform4f	name of the variable then the four floats	Attaches the value to the uniform
setUniformMat4f	name and then the matrix	Attaches the value to the uniform
setUniformMat2f	name and then the matrix	Attaches the value to the uniform

VertexBufferLayout

Variables

Variable Name	**Description**
m_Elements	Stores the elements of the layout
m_Stride	Stores the length of how long each vertex is

Functions

Function Name	**Parameters**	**Description**
calculateOffsetsAndStride		Calculates all the offsets for the elements
getStride		Returns the stride
getElements		Returns the elements
begin		Go between function for the vector m_Elements
end		Go between function for the vector m_Elements

Character

Variables

Variable Name	**Description**
texture	Stores the texture for the character
size	A vector that stores the width and height of the character
bearing	A vector that stores the position relative to the origin
advance	Stores the position of the next character relative to itself

BufferElement

Variables

Variable Name	**Description**
name	Stores the name of the variable input
type	Stores the type of the variable
size	Stores the size of the variable
offset	Stores the offset of its start position
normalized	Stores whether it is normalized

Functions

Function Name	**Parameters**	**Description**
getCompoundCount		Returns the count of elements in each type

VertexBuffer

Variables

Variable Name	**Description**
m_RendererID	Stores the ID of the buffer
m_Offset	Stores the current offset of the buffer of inputting elements
m_BufferSize	Stores the buffer size

Functions

Function Name	Parameters	Description
bind		Binds the buffer
unbind		Unbinds the buffer
clearBufferData		Clears all the data on the buffer
addToBuffer	pointer to the vertices and the size of the vertices	Adds data to the buffer
getBufferSize		Returns the buffer size
canStore	Size of the data	Checks to see if it can store that data size
isEmpty		Returns if the buffer is empty

AnimatedSprite

Variables

Variable Name	Description
sprites	Stores all the sprite IDs
index	Stores the current index of the sprite it is on
texSwapDelay	Stores the delay between the animations
texSwapCount	Stores the counter for the update cycles between the increasing of the index

Functions

Function Name	Parameters	Description
addSprite	sprite ID	Adds the sprite onto the animation
update		Increases the count and goes to next sprite if needed
nextFrame		Increases index by one
setFrame	frame index	Sets the frame to the input
render	Coords and rotation and either width and height or size	Renders the current sprite active

VertexArray

Variables

Variable Name	Description
m_RenderID	Stores the openGL ID

Functions

Function Name	Parameters	Description
addBuffer	VertexBuffer to add and a layout to apply	Binds the vertex buffer to the vertex array and applies a layout to it
bind		Binds the vertex array
unbind		Unbinds the vertex array

Texture

Variables

Variable Name	Description
m_RenderID	Stores openGL ID for the texture
m_FilePath	For debugging purposes stores the filepath of the texture
m_LocalBuffer	Stores the pointer referring to its local buffer where the image is stored
m_Width	Stores the width of the image
m_Height	Stores the height of the image
m_BPP	Stores the bytes per pixel
bufferStorage	Stores what textures are bound to what slot

Functions

Function Name	Parameters	Description
bind	Slot to bind to	Binds image to given slot
unbind		Unbinds the texture
getWidth		Returns the width of the image
getHeight		Returns the height of the image
getTextureInBuffer	texture slot	Returns the texture bound at that slot
getBoundSlot	Texture	Returns the slot that the texture is bound to
clearBufferSlots		Clears all the textures bound

IndexBuffer

Variables

Variable Name	Description
m_RenderID	Stores the openGL ID
m_Count	Stores the amount of squares it can deal with

Functions

Function Name	**Parameters**	**Description**
bind		Binds the index buffer
unbind		Unbinds the index buffer
getCount		Returns count

RenderObject

Variables

Variable Name	**Description**
position	Stores the position of the object
width	Stores the width of the object
height	Stores the height of the object
rotation	Stores the rotation of the object
centered	Stores whether the points are centered or not

Functions

Function Name	**Parameters**	**Description**
getSizeOfVertices		Returns the size of the vertices that is intended
convertToVertices		Returns the object however in a array of 4 vertices

TexturedObject

Variables

Variable Name	**Description**
spriteID	Stores the sprite id of the object

Functions

Function Name	**Parameters**	**Description**
convertToTexturedVertices	texSlot of the sprite	returns textured vertices representation of the object

ColouredObject

Variables

Variable Name	**Description**
colour	Stores a vec4 which represents the colour of the object

Functions

Function Name	**Parameters**	**Description**
convertToColouredVertices		Converts the object to coloured vertices

RenderTextObject

Variables

Variable Name	**Description**
text	Stores the text of the object
scale	Stores the scale of the text

Functions

Function Name	**Parameters**	**Description**
convertCharacterToVertices	character offset of the text position and the texture slot storing the character's texture	Converts the object into TextVertexes

Vertex

Variables

Variable Name	**Description**
position	Stores the position of the vertex

ColouredVertex

Variables

Variable Name	**Description**
colour	Stores the colour of the vertex

TexturedVertex

Variables

Variable Name	**Description**
texCoords	Stores the position on the texture this vertex represents
texID	Stores the texture ID / slot of the texture it represents

TextVertex

Variables

Variable Name	**Description**
colour	Stores the colour of the text

2.7.8 Effects

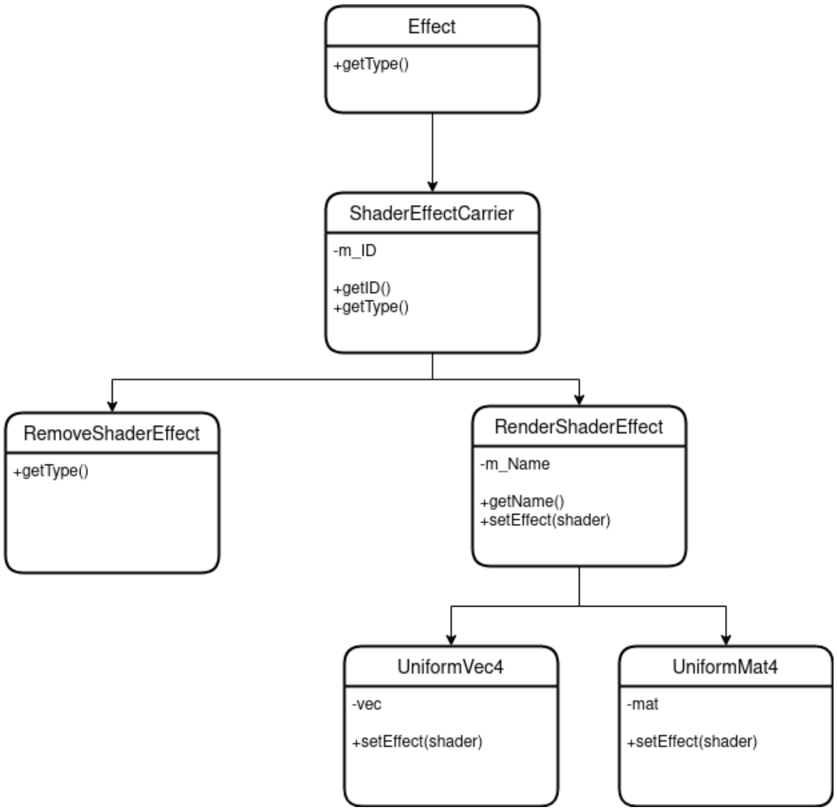


Figure 18: Effect subclasses

Effect

Functions

Function Name	Parameters	Description
getType		Returns the type of the effect

ShaderEffectCarrier

Variables

Variable Name	Description
m_ID	Stores the ID of the effect

Functions

Function Name	Parameters	Description
getID		Returns the ID of the effect

RemoveShaderEffect - Inherits everything from parent classes and only overrides
ShaderEffect

Variables

Variable Name	Description
m_Name	Stores the name of the variable

Functions

Function Name	Parameters	Description
getName		Returns the name of the variable
setEffect	shader to set the effect to	Sets the current effect to the shader given

UniformVec4

Variables

Variable Name	Description
vec	Stores the vector that is passed into the shader

UniformMat4

Variables

Variable Name	Description
mat	Stores the matrix that is passed into the shader

2.7.9 Events

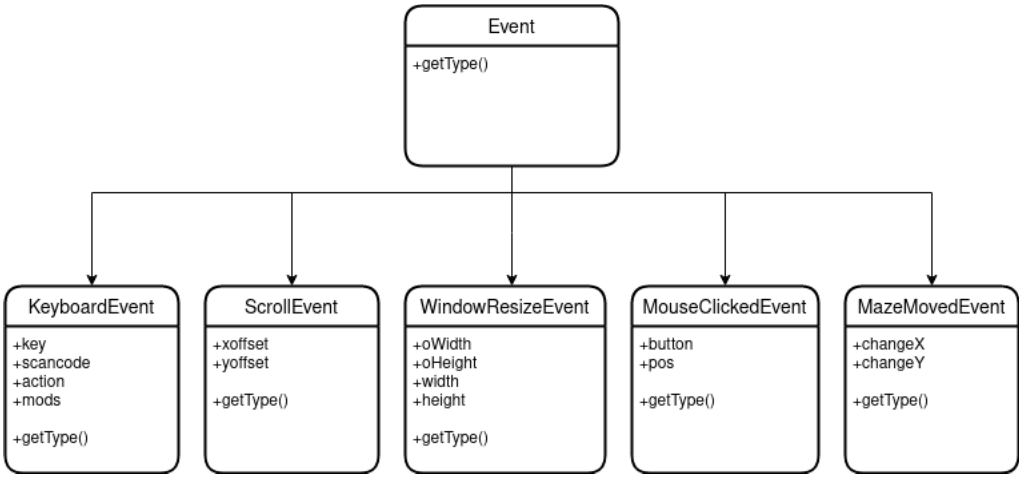


Figure 19: Event subclasses

Event

Functions

Function Name	**Parameters**	**Description**
getType		Returns the type of event

keyboardEvent

Variables

Variable Name	**Description**
key	Stores the key pressed
scancode	Stores the platform-specific scancode
action	Stores the action of the key (Release, press, hold)
mods	Stores the modifier bits

ScrollEvent

Variables

Variable Name	**Description**
xoffset	Stores the change in the x direction
yoffset	Stores the change in the y direction

WindowResizeEvent

Variables

Variable Name	**Description**
oWidth	Stores the width before the transformation
oHeight	Stores the height before the transformation
width	Stores the new width
height	Stores the new height

MouseClickedEvent

Variables

Variable Name	**Description**
button	Stores the button that has been pressed
pos	Stores the position of the mouse

MazeMovedEvent

Variables

Variable Name	**Description**
changeX	Stores the change in X that has happened
changeY	Stores the change in Y that has happened

2.7.10 Other

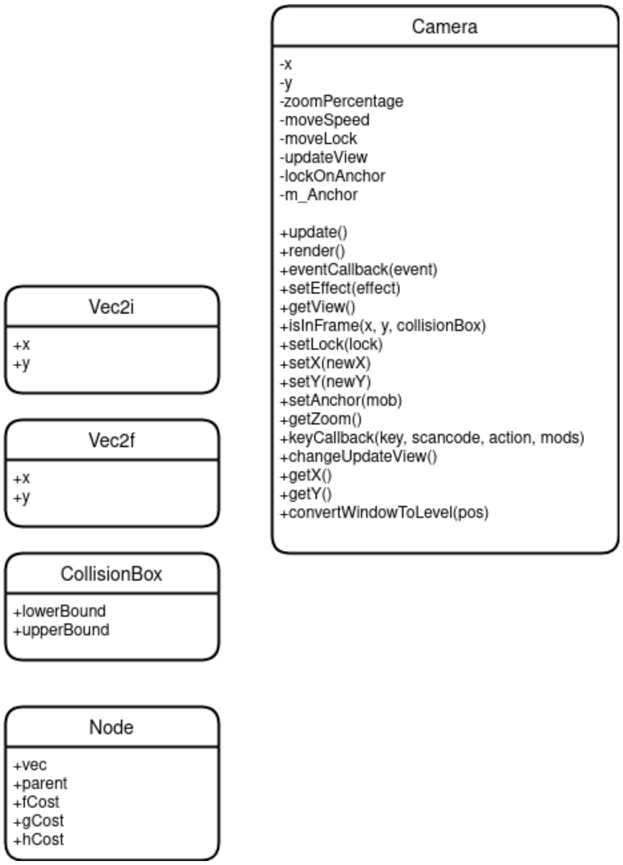


Figure 20: Classes that are for general use

Camera

Variables

Variable Name	Description
x	Stores x position of the camera
y	Stores the y position of the camera
zoomPercentage	Stores the zoom percentage of the objects
moveSpeed	Stores the speed it can move when disconnected from its anchor
moveLock	Stores whether it can move or not
updateView	Stores whether the view effect needs to be updated
lockOnAnchor	Stores whether it needs to be locked on its anchor
m_Anchor	Stores the entity it is locked onto

Functions

Function Name	Parameters	Description
update		Updates the camera position
render		Updates the view effect
eventCallback	event	Deals with current event
setEffect	effect	Allows camera to receive an effect
getView		Returns the view matrix
isInFrame	Position and collision box	Returns whether it will be displayed onscreen
setLock	lock	Sets the lock
setX	new x coord	sets the x value
setY	new y coord	sets the y value
setAnchor	mob	Sets the anchor
getZoom		Returns the zoom
keyCallback	information stored in key event	Deals with a key being pressed
changeUpdateView		changes the updateView variable to true so the view will be updated next render cycle
getX		Returns the x value
getY		Returns the y value
convertWindowToLevel	Position vector	Converts the position into coordinates in the level

Vec2i

Variables

Variable Name	**Description**
x	Stores x position as an int
y	Stores y position as an int

Vec2f

Variables

Variable Name	**Description**
x	Stores x position as an float
y	Stores y position as an float

CollisionBox

Variables

Variable Name	**Description**
lowerBound	Stores the position of the bottom left corner (relative to the objects coordinates)
upperBound	Stores the position of the top right corner (relative to the objects coordinates)

Node

Variables

Variable Name	**Description**
vec	Stores the position of the node (as integer in the grid)
parent	Stores the parent (as a position on the grid)
fCost	The total cost of the node
gCost	The distance from the start node
hCost	The distance from the destination node

2.8 Functions

2.8.1 Control

Function Name	**Parameters**	**Description**
main		First function that is run when the program boots up
gameLoop		Function that controls the game loop and tells the application when to render and update

2.8.2 Utils

Function Name	**Parameters**	**Description**
getIndexOfInsertion	Array which the element will be added to, nodeMap and the next node	Uses binomial search to find the position of where to insert a new element
factorial	num	Returns the result of a factorial
directionToRotation	direction	Converts a direction into radians
distanceBetweenVec2i	start and end positions	Calculates the distance between two vectors using pythagoras
distanceBetweenVec2f	start and end positions	Calculates the distance between two vectors using pythagoras

3 Technical Solution

All of the files here are relative to the "src" folder.

3.1 Main files

Main.cpp

```
#include "Core.h"

#include <chrono>
#include <thread>

#include "glDebug.h"

#include "Camera.h"
#include "Renderer.h"

#include "Application.h"
#include "Log.h"
#include "RandomGen.h"

#include "Layer.h"
#include "Maze.h"

#include "Button.h"

#include "Tile.h"

// This is the game loop that keeps the game running
void gameLoop()
{
#ifdef DEBUG
    ImGuiIO &io = *Application::getImGuiContext();   // Creates an ImGui Interface, if I am debugging
#endif

    int          fps = 0;
    int          ups = 0;
    const double ns  = 1000000000.0f / 60.0f;

    auto   lastTime = std::chrono::high_resolution_clock::now();
    double delta    = 1.0f;

    while(Application::isWindowOpen())
    {
#ifdef DEBUG
        GLCall(glClearColor(1.0f, 1.0f, 1.0f, 1.0f));   // Sets the background to white if I am Debugging as it is
     easier to see if textures are not rendering
#endif
        glClear(GL_COLOR_BUFFER_BIT);   // Resets the screen

        auto now = std::chrono::high_resolution_clock::now();
        delta += (double) std::chrono::duration_cast<std::chrono::nanoseconds>(now - lastTime).count() / ns;
        lastTime = now;
        // Updates and renders the application
        while(delta >= 1)
        {
            Application::update();
            ups++;
            delta--;
        }
        fps++;
        Application::render();

#ifdef DEBUG   // Renders all the ImGui interface to make it easier while debugging
        ImGui_ImplOpenGL3_NewFrame();
        ImGui_ImplGlfw_NewFrame();
        ImGui::NewFrame();

        ImGui::Begin("MazeGame");
        Application::imGuiRender();
        // Shows the framerate of the program
        ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / ImGui::GetIO().Framerate, ImGui::
    GetIO().Framerate);
        ImGui::End();

        ImGui::Render();
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
#endif
```

```
        // Swaps the buffers of the application
        Application::swapBuffers();
        glfwPollEvents();   // Processes all pending events
#ifdef DEBUG
        if(io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
        {
            GLFWwindow *backup_current_context = glfwGetCurrentContext();
            ImGui::UpdatePlatformWindows();
            ImGui::RenderPlatformWindowsDefault();
            glfwMakeContextCurrent(backup_current_context);
        }
#endif
    }
}

int main(void)
{
    // TODO: Add check to see if there was a problem booting up

    Event::init();   // Initialises the events (in Event.h)

    Log::info("Initialised program");

    Application::setupLayers();

    gameLoop();   // Starts the game loop

    return 0;
}
```

../src/MazeGame.cpp

Application.h

```
#pragma once

#include <GLM.h>

#include "Camera.h"
#include "Effect.h"
#include "Layer.h"
#include "Log.h"

#include "Event.h"

class Application
{
  public:
    Application(const Application &) = delete;

    ~Application();

    static Application &get()
    {
        static Application instance;
        return instance;
    }

#ifdef DEBUG
    static ImGuiIO *getImGuiContext()
    {
        return get().getImGuiContextImpl();
    }
    static bool setupImGui() { return get().setupImGuiImpl(); }
    static void imGuiRender() { get().imGuiRenderImpl(); }
#endif

    static void update()
    {
        get().updateImpl();
    }
    static void render() { get().renderImpl(); }

    static void setupLayers() { get().setupLayersImpl(); }
    static void exitGame() { get().setupLayersImpl(); }
    static void startGame() { get().startGameImpl(); }

    static void addLayer(Layer *layer) { get().addLayerImpl(layer); }
    static void addLayer(Layer *layer, int index) { get().addLayerImpl(layer, index); }
    static void addOverlay(Layer *layer) { get().addOverlayImpl(layer); }
    static void removeLayer(int index) { get().removeLayerImpl(index); }
    static void removeLayer(Layer *layer, bool deleteLayer = false) { get().removeLayerImpl(layer, deleteLayer); }

    static void callEvent(const Event::Event &e, bool includeOverlay = false) { get().callEventImpl(e,
    includeOverlay); }
    static void callEventLater(const Event::Event *e) { get().eventBuffer.push_back(e); }
    static void setEffect(Effect::Effect *e, bool includeOverlay = false) { get().setEffectImpl(e, includeOverlay)
    ; }
    static void setOverlayEffect(Effect::Effect *e) { get().setOverlayEffectImpl(e); }

    static void updateWindowSize(int width, int height) { get().updateWindowSizeImpl(width, height); }
    static bool isWindowOpen() { return get().isWindowOpenImpl(); }
    static void swapBuffers() { get().swapBuffersImpl(); }
    static bool isInFrame(float x, float y, CollisionBox box) { return get().isInFrameImpl(x, y, box); }
    static void closeApplication() { get().closeApplicationImpl(); }

    static Camera *  getCamera() { return get().getCameraImpl(); }
    static glm::mat4 getProj() { return get().getProjImpl(); }
    static int       getWidth() { return get().getWidthImpl(); }
    static int       getHeight() { return get().getHeightImpl(); }
    static void *    getWindow() { return get().getWindowImpl(); }

    static bool getIsPaused() { return get().gameIsPaused; }
    static void setIsPaused(bool paused) { get().gameIsPaused = paused; }

  private:
    static Application s_Instance;

    GLFWwindow *window;   // Stores the GLFW winodow
    Camera      camera;

    int       windowWidth, windowHeight;
    glm::mat4 proj;   // Stores the projection mapping for the window

    int                 overlayStart;
    std::vector<Layer *> layers;   // This will store all the layers needed (I don't have to use a vector here as
    I know what is the maximum layers that will be used at one time

    // Used for storing Events until they can be run without causing errors
    std::vector<const Event::Event *> eventBuffer;

    uint16_t projEffectID;

    bool gameIsPaused;

    Application();

#ifdef DEBUG
    ImGuiIO *getImGuiContextImpl();
    bool     setupImGuiImpl();
    void     imGuiRenderImpl();
#endif

    void updateImpl();
    void renderImpl();

    void setupLayersImpl();
    void startGameImpl();
    void addLayerImpl(Layer *layer);
    void addLayerImpl(Layer *layer, int index);
    void addOverlayImpl(Layer *layer);
    void removeLayerImpl(int index);
    void removeLayerImpl(Layer *layer, bool deleteLayer);

    void callEventImpl(const Event::Event &e, bool includeOverlay = false);
    void setEffectImpl(Effect::Effect *e, bool includeOverlay = false);
    void setOverlayEffectImpl(Effect::Effect *e);

    void updateWindowSizeImpl(int width, int height);
    bool isWindowOpenImpl();
    void swapBuffersImpl();
    bool isInFrameImpl(float x, float y, CollisionBox box);
    void closeApplicationImpl();

    Camera *  getCameraImpl();
    glm::mat4 getProjImpl();
    int       getWidthImpl();
    int       getHeightImpl();
    void *    getWindowImpl();
};
```

../src/Application.h

Application.cpp

```
#include "Core.h"

#include <algorithm>
#include <functional>
#include <vector>

#include "Application.h"
#include "Renderer.h"
#include "Sprite.h"
#include "Tile.h"
#include "glDebug.h"

#include "Maze.h"

#include "VertexBufferLayout.h"

#include "GUILayer.h"

#include "Event.h"
#include "Log.h"
#include "MessageManager.h"
#include "ShaderEffectsManager.h"

// SECTION: Initialises
Application::Application()
    : camera(4500.0f, 4500.0f),
      windowWidth(940),
      windowHeight(540),
      proj(glm::ortho(0.0f, (float) windowWidth, 0.0f, (float) windowHeight, -100.0f, 100.0f)),
      overlayStart(0),
      gameIsPaused(false)
{   // This initialises everything
    layers.reserve(2);

    if(!glfwInit())   // Initialises GLFW, and checks it was okay
    {
        Log::critical("GLFW failed to initialise", LOGINFO);   // Logs a critical error
    }

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);   // Sets the openGL version
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 4);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    window = glfwCreateWindow(windowWidth, windowHeight, "MazeGame", NULL, NULL);
    if(!window)   // Checks window is not a nullpointer
    {
        Log::critical("Window seems to be a nullptr, will now shutdown... I do not feel well", LOGINFO);
    }

    glfwMakeContextCurrent(window);   // Makes context and makes it so that the program can only run at 60fps or
    lower (for a more constant framerate)
    glfwSwapInterval(0);

    if(glewInit() != GLEW_OK)   // Initialises GLEW
    {
        Log::critical("GLEW is not OK please send help", LOGINFO);
    }

    // Logs the open GL version (from the graphics card)
    Log::variable<const GLubyte *>("GL version", glGetString(GL_VERSION));

    // Enables the default blending
    GLCall(glEnable(GL_BLEND));
    GLCall(glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA));

    Sprite::init();   // Initialises all the sprites
}

Application::~Application()   // Terminates everything
{
    Log::info("Shutting down");

    // Deletes all the layers (as they are allocated on the heap)
    for(int i = 0; i < layers.size(); i++)
    {
```

```cpp
        if(layers[i])
            delete layers[i];
    }

#ifdef DEBUG
    ImGui_ImplOpenGL3_Shutdown();   // Shuts down ImGui
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();   // Destoys ImGui Context
#endif

    glfwTerminate();   // Terminates glfw
}

#ifdef DEBUG
bool Application::setupImGuiImpl()   // Sets up ImGui
{
    ImGui::CreateContext();   // Creates ImGui context
    ImGuiIO &io = ImGui::GetIO();
    // io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
    io.ConfigFlags |= ImGuiConfigFlags_DockingEnable;
    io.ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;

    ImGui::StyleColorsDark();
    ImGuiStyle &style = ImGui::GetStyle();
    if(io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
    {
        style.WindowRounding              = 0.0f;
        style.Colors[ImGuiCol_WindowBg].w = 1.0f;
    }
    bool output1, output2;
    output1 = ImGui_ImplGlfw_InitForOpenGL(window, true);
    output2 = ImGui_ImplOpenGL3_Init("#version 150");
    if(output1 && output2)
        return true;
    else
    {
        Log::critical("ImGUI failed", LOGINFO);
        return false;
    }
}

ImGuiIO *Application::getImGuiContextImpl()
{
    ImGui::CreateContext();   // Creates ImGui context
    ImGuiIO &io = ImGui::GetIO();
    // io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
    io.ConfigFlags |= ImGuiConfigFlags_DockingEnable;
    io.ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;

    ImGui::StyleColorsDark();
    ImGuiStyle &style = ImGui::GetStyle();
    if(io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
    {
        style.WindowRounding              = 0.0f;
        style.Colors[ImGuiCol_WindowBg].w = 1.0f;
    }
    bool output1, output2;
    output1 = ImGui_ImplGlfw_InitForOpenGL(window, true);
    output2 = ImGui_ImplOpenGL3_Init("#version 150");
    if(output1 && output2)
        return &io;
    else
    {
        Log::critical("ImGUI failed while creating the context", LOGINFO);
        return nullptr;
    }
}
#endif

void Application::updateImpl()   // Updates all the layers
{
    Event::update();
    if(projEffectID == 0)
        updateWindowSizeImpl(windowWidth, windowHeight);
    for(int i = layers.size() - 1; i > -1; i--)
    {
        layers[i]->update();
        if(gameIsPaused && i == overlayStart)
            break;
    }
    camera.update();

    MessageManager::update();

    for(const Event::Event *e : eventBuffer)
    {
        callEvent(*e, true);
        delete e;
    }
    eventBuffer.clear();
}

void Application::renderImpl()   // Renders all the layers
{
    camera.render();
    for(int i = 0; i < layers.size(); i++)
    {
        layers[i]->render();
        Render::render(layers[i]->getShaderEffects());
    }

    // TODO:: Make a layer for this or do something clever
    MessageManager::render();
    std::vector<uint16_t> temp;
    Render::render(temp);
}

#ifdef DEBUG
void Application::imGuiRenderImpl()   // Renders ImGui in all the layers
{
    for(int i = 0; i < layers.size(); i++)
        layers[i]->imGuiRender();
    camera.imGuiRender();
}
#endif
// !SECTION

// SECTION: Layers

void Application::setupLayersImpl()
{
    gameIsPaused = true;
    // TODO: Put this in a separate function
    for(Layer *layer : layers)
        delete layer;
    layers.clear();
    overlayStart = 0;
    camera.clearAnchor();

    addOverlayImpl(new GUILayer(GUILayer::Type::MainMenu, nullptr));
    Effect::ShaderEffectsManager::updateShaderEffects();
}

void Application::startGameImpl()
{
    gameIsPaused = false;
    for(Layer *layer : layers)
        delete layer;
    layers.clear();
    overlayStart = 0;

    Maze *maze = new Maze();
    maze->generate();   // Generates the maze
    addLayer(maze);     // Adds it to the layers

    Effect::ShaderEffectsManager::updateShaderEffects();
}

void Application::addLayerImpl(Layer *layer)   // Inserts a layer before the background
{
    layers.insert(layers.begin() + overlayStart, layer);
    overlayStart++;
}

void Application::addLayerImpl(Layer *layer, int index)   // Adds layer at a given index
{
    layers.insert(layers.begin() + index, layer);
    overlayStart++;
}
```

```
void Application::addOverlayImpl(Layer *layer)   // Adds an overlay to the layer stack, meaning it is appended to
    the end of the vector
{
    layers.push_back(layer);
}

void Application::removeLayerImpl(int index)   // Removes layer
{
    layers.erase(layers.begin() + index);
}

void Application::removeLayerImpl(Layer *layer, bool deleteLayer)
{
    std::vector<Layer *>::iterator index = std::find(layers.begin(), layers.end(), layer);
    if(index != layers.end())
    {
        if(deleteLayer)
            delete layers[index - layers.begin()];
        layers.erase(index);
    }
    else
        Log::warning("Cannot find layer to remove!");
}
// !SECTION

// SECTION: Events & Effects
void Application::callEventImpl(const Event::Event &e, bool includeOverlay)   // Sends event through the layers
{
    // TODO: Make this multithreading
    if(camera.eventCallback(e))
        return;

    int startVal;
    if(includeOverlay)
        startVal = layers.size();
    else
    {
        if(gameIsPaused && !e.ignoreIfPaused())
            return;
        startVal = overlayStart;
    }

    for(int i = startVal - 1; i > -1; i--)
    {
        if(layers[i])
        {
            if(layers[i]->eventCallback(e))
                break;
        }

        if(gameIsPaused && i == overlayStart && !e.ignoreIfPaused())
            break;
    }
}

void Application::setEffectImpl(Effect::Effect *e, bool includeOverlay)   // Sends an effect through the layers
{
    if(e->getType() == Effect::Effect::Type::removeShaderEffect)
    {
        Effect::RemoveShaderEffect *ne = static_cast<Effect::RemoveShaderEffect *>(e);
        if(ne->getID() == projEffectID)
        {
            Log::warning("Deleting projection effect!");
            projEffectID = 0;
        }
        else if(ne->getID() > projEffectID)
            projEffectID--;
    }
    int endVal;
    if(includeOverlay)
        endVal = layers.size();
    else
        endVal = overlayStart;

    for(int i = 0; i < endVal; i++)
        layers[i]->setEffect(e);
}

void Application::setOverlayEffectImpl(Effect::Effect *e)
{
    for(int i = overlayStart; i < layers.size(); i++)
        layers[i]->setEffect(e);
}

// !SECTION

// SECTION: Window stuff
void Application::updateWindowSizeImpl(int width, int height)   // updates the window size and projection matrix
{
    windowWidth  = width;
    windowHeight = height;
    proj         = glm::ortho(0.0f, (float) width, 0.0f, (float) height, -100.0f, 100.0f);
    if(projEffectID == 0)
    {
        std::string name = "u_MVP";
        projEffectID     = Effect::ShaderEffectsManager::sendOverlayEffect(name, proj);
    }
    else
    {
        Effect::UniformMat4 *e = static_cast<Effect::UniformMat4 *>(Effect::ShaderEffectsManager::getShaderEffect(
    projEffectID));   // TODO: Change this to a dynamic cast or make a function for it
        e->setMat(proj);
    }
}

bool Application::isWindowOpenImpl()   // Returns if the window is still open
{
    return !glfwWindowShouldClose(window);
}

void Application::swapBuffersImpl()   // Swaps the buffers
{
    glfwSwapBuffers(window);
}

bool Application::isInFrameImpl(float x, float y, CollisionBox box)
{
    return camera.isInFrame(x, y, box);
}

void Application::closeApplicationImpl()
{
    glfwDestroyWindow(window);
}

// !SECTION

// SECTION: Getters
int        Application::getWidthImpl() { return windowWidth; }
int        Application::getHeightImpl() { return windowHeight; }
void *     Application::getWindowImpl() { return window; }
Camera *   Application::getCameraImpl() { return &camera; }
glm::mat4  Application::getProjImpl() { return proj; }
// !SECTION
```

../src/Application.cpp

3.2 Utils

3.2.1 General Utils

utils/AStarUtils.h

```
#pragma once

#include <functional>
#include <vector>
#include <algorithm>
#include <array>

// #include <filesystem>
// #include <fstream>

#include "Utils.h"

#define X_MAX 1400
#define Y_MAX 1400

#define X_STEP 50
```

```
#define Y_STEP 50
struct Node
{
    Vec2i parent;
    float fCost, gCost, hCost;

    Node()
        : parent({-1, -1}), fCost(-1), gCost(-1), hCost(-1) {}
};

inline bool operator<(const Node &lhs, const Node &rhs)
{
    return lhs.fCost < rhs.fCost;
}
inline bool operator>(const Node &lhs, const Node &rhs)
{
    return lhs.fCost > rhs.fCost;
}
inline bool operator==(const Node &lhs, const Node &rhs)
{
    return lhs.fCost == rhs.fCost;
}
inline bool operator!=(const Node &lhs, const Node &rhs)
{
    return lhs.fCost == rhs.fCost;
}

// SORTING HIGHEST TO LOWEST
template <size_t ySize, size_t xSize>
inline int getIndexOfInsertion(std::vector<Vec2i> positions, std::array<std::array<Node, ySize>, xSize> &nodeMap,
    Vec2i nextPos)
{
    if(positions.size() == 0)
        return 0;
    if(positions.size() == 1)
    {
        if(nodeMap[positions[0].x][positions[0].y] > nodeMap[nextPos.x][nextPos.y])
            return 1;
        else
            return 0;
    }
    int startSub = 0;
    int endSub   = positions.size();
    int index    = positions.size() / 2;

    Node &node = nodeMap[nextPos.x][nextPos.y];

    auto getNode = [&nodeMap, &positions](int index) -> Node & {
        return nodeMap[positions[index].x][positions[index].y];
    };

    while(startSub != endSub && startSub < endSub)
    {
        if(index + 1 >= positions.size())
            index = positions.size() - 2;

        Node &thisNode = getNode(index);
        Node &nextNode = getNode(index + 1);

        if(thisNode == node || (thisNode > node && nextNode < node))
            return index + 1;
        else if(nextNode == node)
            return index + 2;
        else if(thisNode > node && nextNode > node)
            startSub = index + 2;
        else if(thisNode < node && nextNode < node)
            endSub = index;
        else
        {
            Log::critical("Node vector is not sorted correctly!", LOGINFO);
            return -1;
        }

        index = (startSub + endSub) / 2;
    }

    return index;
}

// static uint32_t fileCount = 1;
```

```cpp
static bool hasWarned = false;

// NOTE: This is meant to deal with a number of situations in which to use A star, so if the grid is made to only be a portion of the map, the conversion must happen beforehand
template <size_t width, size_t height, size_t numOfPoints>
inline std::vector<Vec2f> *aStarAlgorithm(Vec2i startPos, Vec2i destPos, CollisionBox box, std::array<Vec2i, numOfPoints> &offsets, std::function<bool(int, int, int, int, CollisionBox)> &collisionDetection, std::function<Vec2f(Vec2i)> &convert, int pathLimit)
{

    std::vector<Vec2f> *path = new std::vector<Vec2f>();

    if(startPos.x != destPos.x || startPos.y != destPos.y)
    {
        // std::stringstream log;
        // log << "NOTE START" <<  startPos << "\n";
        // log << "NOTE DESTINATION" <<  destPos << "\n";

        if(collisionDetection(destPos.x, destPos.y, 0, 0, box))
        {
            // Log::error("Cannot reach destination!", LOGINFO);
            std::vector<Vec2f> *path = new std::vector<Vec2f>();
            path->push_back(convert(destPos));
            return path;
        }
        std::vector<Vec2i> openList;
        openList.reserve(width * height);

        bool closedList[width][height];
        memset(closedList, false, sizeof(closedList));

        std::array<std::array<Node, height>, width> nodeMap;

        {
            float hCost                              = distBetweenVec2i(startPos, destPos);
            nodeMap[startPos.x][startPos.y].parent = {-1, -1};
            nodeMap[startPos.x][startPos.y].fCost  = hCost;
            nodeMap[startPos.x][startPos.y].gCost  = 0.0f;
            nodeMap[startPos.x][startPos.y].hCost  = hCost;

            openList.push_back(startPos);
        }

        auto skipThisPos = [&](Vec2i nextPos, Vec2i offset) -> bool {
            if(nextPos.x >= width || nextPos.x < 0 || nextPos.y >= height || nextPos.y < 0)
            {
                // log << "REJECTED: out of bounds\n";
                return true;
            }

            if(collisionDetection(nextPos.x - offset.x, nextPos.y - offset.y, offset.x, offset.y, box))
            {
                // log << "REJECTED: collision detected\n";
                return true;
            }

            if(closedList[nextPos.x][nextPos.y])
            {
                // log << "REJECTED: Node has been closed\n";
                return true;
            }
            return false;
        };

        while(openList.back() != destPos)
        {
            Vec2i currentPos  = openList.back();
            openList.pop_back();
            // log << "Current: " <<  currentPos << "\n";

            Node *currentNode = &nodeMap[currentPos.x][currentPos.y];
            closedList[currentPos.x][currentPos.y] = true;
            for(Vec2i offset : offsets)
            {
                Vec2i nextPos = {currentPos.x + offset.x, currentPos.y + offset.y};
                // log << "NextPos: " <<  nextPos << "\n";

                if(skipThisPos(nextPos, offset))
                    continue;
            float gCost = currentNode->gCost + distBetweenVec2i(currentPos, nextPos);
            float hCost = distBetweenVec2i(nextPos, destPos);
            float fCost = gCost + hCost;
            if(nodeMap[nextPos.x][nextPos.y].fCost != -1)
            {
                // log << "Node already explored\n";
                if(fCost >= nodeMap[nextPos.x][nextPos.y].fCost)
                {
                    // log << "REJECTED: has a higher fCost!\n";
                    continue;
                }
                nodeMap[nextPos.x][nextPos.y].parent = currentPos;
                nodeMap[nextPos.x][nextPos.y].fCost  = fCost;
                nodeMap[nextPos.x][nextPos.y].gCost  = gCost;
                nodeMap[nextPos.x][nextPos.y].hCost  = hCost;
                openList.erase(std::find(openList.begin(), openList.end(), nextPos));
            }
            else
            {
                // nodeMap[currentPos.x + dirVec.x][currentPos.y + dirVec.y].vec    = nextPos;
                nodeMap[nextPos.x][nextPos.y].parent = currentPos;
                nodeMap[nextPos.x][nextPos.y].fCost  = fCost;
                nodeMap[nextPos.x][nextPos.y].gCost  = gCost;
                nodeMap[nextPos.x][nextPos.y].hCost  = hCost;
            }

            int insertIndex = getIndexOfInsertion(openList, nodeMap, nextPos);

            openList.insert(openList.begin() + insertIndex, nextPos);
            // log << "Inserted node to list at: " << insertIndex << "\n";
        }
        if(openList.size() == 0)
        {
            if(!hasWarned)
            {
                // Log::warning("Cannot find route to destination");
                hasWarned = true;
                // std::stringstream filenameStream;
                // filenameStream << "A Star Log " << fileCount << ".log";
                // fileCount++;

                // std::ofstream file(filenameStream.str(), std::ios_base::app);   // The std::ios_base::app
allows it to write at end
                // file << log.str() << "Vector size is 0";
                // file.close();
            }
            std::vector<Vec2f> *path = new std::vector<Vec2f>();
            path->push_back(convert(destPos));
            return path;
        }
        else if(openList.size() > width * height)
        {
            Log::critical("Too many nodes in the open list! Something is wrong!", LOGINFO);
            std::vector<Vec2f> *path = new std::vector<Vec2f>();
            path->push_back(convert(destPos));
            return path;
        }
    }

    Vec2i currentPos = openList.back();
    while(currentPos != startPos)
    {
        Node *currentNode = &nodeMap[currentPos.x][currentPos.y];
        path->push_back(convert(currentPos));
        currentPos = currentNode->parent;
        if(path->size() > pathLimit)
        {
            Log::info("Path too long, cancelling");
            path->clear();
            path->push_back(convert(destPos));
            return path;
        }
    }
}
else
    path->push_back(convert(destPos));

if(hasWarned)
    hasWarned = false;
    return path;
}
```

../src/utils/AStarUtils.h

utils/Utils.h

```
#pragma once

enum Direction
{
    north = 0,
    south,
    east,
    west
};

enum class RoomType
{
    Empty,
    Chest,
    Trap,
    Enemy,
    NPC,
    Exit
};

enum InGameGUILayer
{
    overlay = 0,
    playerInventory,
    chestInventory,
    npcInventory,
    npcInteraction,
    exitMenu,
    playerDeath,
    playerWin
};

enum GUIInventoryIDCode
{
    none,
    inventory,
    weapons
};

struct Vec2i
{
    int x, y;
    // Vec2i(): x(0), y(0) {}
};

inline bool operator!=(const Vec2i &lhs, const Vec2i &rhs)
{
    return lhs.x != rhs.x || lhs.y != rhs.y;
}

inline bool operator==(const Vec2i &lhs, const Vec2i &rhs)
{
    return lhs.x == rhs.x && lhs.y == rhs.y;
}

inline std::ostream &operator<<(std::ostream &out, const Vec2i &data)
{
    out << "(" << data.x << ", " << data.y << ")";
    return out;
}

struct Vec2f
{
    float x, y;
    // Vec2f(): x(0.0f), y(0.0f) {}
    // Vec2f(float x, float y): x(x), y(y) {}
};

inline bool operator!=(const Vec2f &lhs, const Vec2f &rhs)
{
    return lhs.x != rhs.x || lhs.y != rhs.y;
}

inline bool operator==(const Vec2f &lhs, const Vec2f &rhs)
```

```
{
    return lhs.x == rhs.x && lhs.y == rhs.y;
}

inline Vec2f operator+(const Vec2f &lhs, const Vec2f &rhs)
{
    return {lhs.x + rhs.x, lhs.y + rhs.y};
}
inline Vec2f operator+(const Vec2f &lhs, const float &rhs)
{
    return {lhs.x + rhs, lhs.y + rhs};
}
inline Vec2f operator-(const Vec2f &lhs, const Vec2f &rhs)
{
    return {lhs.x - rhs.x, lhs.y - rhs.y};
}
inline Vec2f operator-(const Vec2f &lhs, const float &rhs)
{
    return {lhs.x - rhs, lhs.y - rhs};
}

inline std::ostream &operator<<(std::ostream &out, const Vec2f &data)
{
    out << "(" << data.x << ", " << data.y << ")";
    return out;
}

struct CollisionBox
{
    Vec2f lowerBound, upperBound;
};   // Lowerbound is the Bottom Left, upperbound is the top right



uint32_t factorial(int num);
float directionToRotation(Direction dir);

// TODO: Change this into a template
float distBetweenVec2i(const Vec2i &start, const Vec2i &end);
float distBetweenVec2f(const Vec2f &start, const Vec2f &end);

bool doesPointIntersectWithBox(Vec2f point, Vec2f pos, CollisionBox box);
bool doesBoxIntersectWithBox(Vec2f pos1, CollisionBox box1, Vec2f pos2, CollisionBox box2);
```

../src/utils/Utils.h

Utils.cpp

```
#include "Utils.h"

#define _USE_MATH_DEFINES
#include <math.h>

#include "Level.h"

uint32_t factorial(int num)
{
    uint32_t output = 1;
    for(uint16_t i = 1; i <= num; i++)
        output *= i;
    return output;
}

float distBetweenVec2f(const Vec2f &start, const Vec2f &end)
{
    float xDist = end.x - start.x;
    float yDist = end.y - start.y;

    return std::sqrt(xDist * xDist + yDist * yDist);   // Uses Pythagorus
}

float distBetweenVec2i(const Vec2i &start, const Vec2i &end)
{
    float xDist = end.x - start.x;
    float yDist = end.y - start.y;

    return std::sqrt(xDist * xDist + yDist * yDist);   // Uses Pythagorus
}

float directionToRotation(Direction dir)
{
    if(dir == Direction::north)
        return 0.0f;
    else if(dir == Direction::south)
        return M_PI;
    else if(dir == Direction::east)
        return M_PI_2;
    else
        return 3.0f * M_PI_2;
}

bool doesPointIntersectWithBox(Vec2f point, Vec2f pos, CollisionBox box)
{
    float lowerX = pos.x + box.lowerBound.x;
    float lowerY = pos.y + box.lowerBound.y;
    float upperX = pos.x + box.upperBound.x;
    float upperY = pos.y + box.upperBound.y;

    return point.x > lowerX && point.y > lowerY && point.x < upperX && point.y < upperY;
}

bool doesBoxIntersectWithBox(Vec2f pos1, CollisionBox box1, Vec2f pos2, CollisionBox box2)
{
    return !(pos1.x + box1.upperBound.x < pos2.x + box2.lowerBound.x || pos1.x + box1.lowerBound.x > pos2.x + box2.upperBound.x || pos1.y + box1.upperBound.y < pos2.y + box2.lowerBound.y || pos1.y + box1.lowerBound.y > pos2.y + box2.upperBound.y);
}
```

../src/utils/Utils.cpp

utils/RoomColours.h

```
#pragma once

// These are the values of the red value of the colour
#define WALL_COLOUR       0xae
#define FLOOR_COLOUR      0xff
#define CORNER_OUT_COLOUR 0xcb
#define CORNER_IN_COLOUR  0x4d
#define CHEST_COLOUR      0x45
#define DUD_CHEST_COLOUR  0x50
#define TRAP_COLOUR       0x76
#define TRAPDOOR_COLOUR   0x99
```

../src/utils/RoomColours.h

3.2.2 Camera

utils/Camera.h

```
#pragma once

#include <GLM.h>

#include "Layer.h"
#include "Mob.h"

class Camera
{
  private:
    float x, y;
    float zoomPercentage;
    float moveSpeed;
    bool  moveLock;
    bool  updateView;
    bool  lockOnAnchor;
    Mob * m_Anchor;
    uint16_t m_ZoomEffectID;
    uint16_t m_PositionEffectID;

  public:
    Camera();
    Camera(float x, float y);

    void update();
    void render();
#ifdef DEBUG
    void imGuiRender();
#endif

    bool eventCallback(const Event::Event &e);
    bool setEffect(const Effect::Effect &e);

    // TODO: Sort this out
    uint16_t  getPositionEffectID();
    uint16_t  getZoomEffectID();
    void      setShaderEffects();
    void      updatePositionEffect();
    void      updateZoomEffect();
    bool      isInFrame(float x, float y, CollisionBox &box);
    void      setLock(bool locked);
    void      setX(float newX);
    void      setY(float newY);
    void      setAnchor(Mob *e);
    void      clearAnchor();
    float     getZoom();
    void      keyCallback(int key, int scancode, int action, int mods);
    void      changeUpdateView();

    float getX() { return x; }
    float getY() { return y; }

    Vec2f convertWindowToLevel(Vec2f inp);
};
```

../src/utils/Camera.h

utils/Camera.cpp

```
#include "Camera.h"

#include <GLFW/glfw3.h>
#include "ImGui.h"

#include "Application.h"
#include "KeyDefinitions.h"
#include "Log.h"
#include "ShaderEffectsManager.h"

Camera::Camera()
    : x(0.0f), y(0.0f), zoomPercentage(1.0f), moveSpeed(10.0f), moveLock(false), updateView(true), lockOnAnchor(
    false), m_Anchor(nullptr), m_ZoomEffectID(0), m_PositionEffectID(0)
{
    Log::info("Initialised Camera");
}

Camera::Camera(float x, float y)
    : x(x), y(y), zoomPercentage(1.0f), moveSpeed(10.0f), moveLock(false), updateView(true), lockOnAnchor(false),
    m_Anchor(nullptr), m_ZoomEffectID(0), m_PositionEffectID(0)
{
}

void Camera::update()
{
    if(m_Anchor && lockOnAnchor)
    {
        if(m_Anchor->getX() != x || m_Anchor->getY() != y || updateView)
        {
            updateView = true;
            x          = m_Anchor->getX();
            y          = m_Anchor->getY();
        }
    }
    else
    {
        if(Event::isKeyPressed(GLFW_KEY_W) || Event::isKeyPressed(GLFW_KEY_UP))
        {
            updateView = true;
            y += moveSpeed;
        }
        if(Event::isKeyPressed(GLFW_KEY_S) || Event::isKeyPressed(GLFW_KEY_DOWN))
        {
            updateView = true;
            y -= moveSpeed;
        }
        if(Event::isKeyPressed(GLFW_KEY_A) || Event::isKeyPressed(GLFW_KEY_LEFT))
        {
            updateView = true;
            x -= moveSpeed;
        }
        if(Event::isKeyPressed(GLFW_KEY_D) || Event::isKeyPressed(GLFW_KEY_RIGHT))
        {
            updateView = true;
            x += moveSpeed;
        }
    }
}

void Camera::render()
{
    if(updateView)
    {
        updatePositionEffect();
        updateView = false;
    }
}

#ifdef DEBUG
void Camera::imGuiRender()
{
    ImGui::SliderFloat("Speed", &moveSpeed, 0.0f, 20.0f);
    ImGui::SliderFloat("X", &x, -300.0f, 7700.0f);
    ImGui::SliderFloat("Y", &y, -300.0f, 7700.0f);
    int before = zoomPercentage;
    ImGui::SliderFloat("Zoom", &zoomPercentage, 0.05f, 2.0f);
    if(before != zoomPercentage)
    {
        updateZoomEffect();
    }
    if(ImGui::Checkbox("Camera Lock", &lockOnAnchor))
    {
        m_Anchor->setIsInControl(lockOnAnchor);
        updateView = true;
    }
}
#endif

bool Camera::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::scroll)
    {
        const Event::ScrollEvent &ne = static_cast<const Event::ScrollEvent &>(e);

        if(zoomPercentage == 0.25f && ne.yoffset < 0.0f)
            return false;

        float oldZoom = zoomPercentage;
        zoomPercentage += (ne.yoffset * 0.02f) * (zoomPercentage / 0.20f);

        if(zoomPercentage < 0.25f)
            zoomPercentage = 0.25f;

        updateZoomEffect();

        updateView = true;

        return false;
    }
    else if(e.getType() == Event::EventType::windowResize)
    {
        const Event::WindowResizeEvent &ne = static_cast<const Event::WindowResizeEvent &>(e);

        updateView = true;
        return false;
    }
    else if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        changeUpdateView();
    }
    return false;
}

bool Camera::setEffect(const Effect::Effect &e)
{
    return false;
}

uint16_t Camera::getPositionEffectID()
{
    if(m_PositionEffectID == 0)
        setShaderEffects();
    return m_PositionEffectID;
```

```cpp
}
uint16_t Camera::getZoomEffectID()
{
    if(m_ZoomEffectID == 0)
        setShaderEffects();
    return m_ZoomEffectID;
}
void Camera::setShaderEffects()
{
    {
        std::string name   = "u_MVP";
        m_PositionEffectID = Effect::ShaderEffectsManager::sendShaderEffect(name, Application::getProj() * glm::
    translate(glm::mat4(1.0f), glm::vec3(Application::getWidth() / 2 - x * zoomPercentage, Application::getHeight
    () / 2 - y * zoomPercentage, 0.0f)));
    }

    {
        std::string name = "u_Zoom";
        m_ZoomEffectID   = Effect::ShaderEffectsManager::sendShaderEffect(name, glm::vec4(zoomPercentage,
    zoomPercentage, 1.0f, 1.0f));
    }
}

void Camera::updatePositionEffect()
{
    if(m_PositionEffectID == 0)
        setShaderEffects();

    Effect::UniformMat4 *effect = static_cast<Effect::UniformMat4 *>(Effect::ShaderEffectsManager::getShaderEffect
    (m_PositionEffectID));
    effect->setMat(Application::getProj() * glm::translate(glm::mat4(1.0f), glm::vec3(Application::getWidth() / 2
    - x * zoomPercentage, Application::getHeight() / 2 - y * zoomPercentage, 0.0f)));
}
void Camera::updateZoomEffect()
{
    if(m_ZoomEffectID == 0)
        setShaderEffects();

    Effect::UniformVec4 *effect = dynamic_cast<Effect::UniformVec4 *>(Effect::ShaderEffectsManager::
    getShaderEffect(m_ZoomEffectID));
    if(effect)
        effect->setVec(glm::vec4(zoomPercentage, zoomPercentage, 1.0f, 1.0f));
    else
        Log::error("Incorrect effect ID given!", LOGINFO);
}

bool Camera::isInFrame(float objX, float objY, CollisionBox &box)
{   // TODO: Try and make the buffer of 1 tiles slightly smaller :D
    return objX + box.upperBound.x + TILE_SIZE > x - Application::getWidth() / (zoomPercentage * 2) && objX + box.
    lowerBound.x - TILE_SIZE <= x + Application::getWidth() / (zoomPercentage * 2) && objY + box.upperBound.y +
    TILE_SIZE > y - Application::getHeight() / (zoomPercentage * 2) && objY + box.lowerBound.y - TILE_SIZE <= y +
    Application::getHeight() / (zoomPercentage * 2);
}

void Camera::setLock(bool locked)
{
    moveLock = locked;
}

void Camera::setX(float newX)
{
    x = newX;
}

void Camera::setY(float newY)
{
    y = newY;
}

void Camera::setAnchor(Mob *e)
{
    lockOnAnchor = true;
    m_Anchor     = e;
    m_Anchor->setIsInControl(true);
}

void Camera::clearAnchor()
{
    lockOnAnchor = false;
    m_Anchor     = nullptr;
```

```cpp
}

float Camera::getZoom()
{
    return zoomPercentage;
}

void Camera::changeUpdateView()
{
    updateView = true;
}

Vec2f Camera::convertWindowToLevel(Vec2f inp)
{
    float nx = x + (2 * inp.x - Application::getWidth()) / (2 * zoomPercentage);
    float ny = y + (2 * inp.y - Application::getHeight()) / (2 * zoomPercentage);
    return {nx, ny};
}
```

../src/utils/Camera.cpp

3.2.3 Containers

utils/Container.h

```cpp
#pragma once

#include <vector>

#include "Item.h"
#include "Log.h"

template <typename T>
class Container : public std::vector<T>
{
    uint16_t maxSize;

  public:
    Container(uint16_t max)
        : maxSize(max)
    {
        std::vector<T>::reserve(maxSize);
    }
    ~Container()
    {
    }

    bool     isFull() { return std::vector<T>::size() == maxSize; }
    void     setMaxSize(int maxSize) { maxSize = maxSize; }
    uint16_t getMaxSize() { return maxSize; }

    void push_back(const T &element)
    {
        if(std::vector<T>::size() < maxSize)
            std::vector<T>::push_back(element);
        else
        {
            Log::warning("Cannot store anymore!");
        }
    }

    void insert(typename std::vector<T>::iterator it, const T &element)
    {
        if(std::vector<T>::size() < maxSize)
            std::vector<T>::insert(it, element);
        else
        {
            Log::warning("Cannot store anymore!");
        }
    }
};

class IContainer
{
  public:
    enum Type
    {
        item,
        weapon
    };
  public:
    IContainer() {}
    virtual ~IContainer() {}

    virtual Type     getType() const          = 0;
    virtual Item *   getItem(int index) const = 0;
    virtual uint16_t size()                   = 0;
};
```

../src/utils/Container.h

utils/ItemContainer.h

```
#pragma once

#include "Container.h"

class ItemContainer : public IContainer, public Container<Item *>
{
  public:
    ItemContainer(uint16_t max)
        : IContainer(), Container<Item *>(max) {}

    virtual ~ItemContainer() override
    {
        for(Item *item : (*this))
            delete item;
    }

    virtual Type getType() const override { return IContainer::Type::item; }

    virtual Item *getItem(int index) const override
    {
        return Container<Item *>::operator[](index);
    }

    virtual uint16_t size()
    {
        return Container<Item *>::size();
    }
};
```

../src/utils/ItemContainer.h

utils/WeaponContainer.h

```
#pragma once

#include "Container.h"

#include "Weapon.h"

class WeaponContainer : public IContainer, public Container<Weapon *>
{
  public:
    WeaponContainer(uint16_t max)
        : IContainer(), Container<Weapon *>(max) {}

    virtual ~WeaponContainer() override
    {
        for(Weapon *weapon : (*this))
            delete weapon;
    }

    virtual Type getType() const override { return IContainer::Type::weapon; }

    virtual Item *getItem(int index) const override
    {
        return static_cast<Item *>(Container<Weapon *>::operator[](index));
    }

    virtual uint16_t size()
    {
        return Container<Weapon *>::size();
    }
};
```

../src/utils/WeaponContainer.h

3.2.4 Event

utils/Event.h

```
#pragma once

#include "Utils.h"

#include "Container.h"

class Mob;

namespace Event
{
    void  init();
    void  update();
    bool isKeyPressed(int key);
    Vec2f getMousePos();

    enum MouseButton
    {
        leftButton = 0,
        rightButton,
        middleButton,
        button4,
        button5,
        button6,
        button7,
        button8,
    };

    enum EventType
    {
        keyInput,
        scroll,
        mouseMove,
        mouseClicked,
        windowResize,
        mazeMovedEvent,
        showAltTileEvent,
        itemTransfer,
        changeGUILayer,
        chestOpened,
        playerResponse,
        mobDied
    };

    struct Event
    {
        virtual EventType const getType() const = 0;
        virtual bool            ignoreIfPaused() const = 0;
    };

    struct KeyboardEvent : Event
    {
        int key, scancode, action, mods;

        KeyboardEvent(int key, int scancode, int action, int mods)
            : key(key), scancode(scancode), action(action), mods(mods) {}
        virtual EventType const getType() const override { return EventType::keyInput; }
        virtual bool            ignoreIfPaused() const override { return false; }
    };

    struct ScrollEvent : Event
    {
        double xoffset, yoffset;

        ScrollEvent(double xoffset, double yoffset)
            : xoffset(xoffset), yoffset(yoffset) {}
        virtual EventType const getType() const override { return EventType::scroll; }
        virtual bool            ignoreIfPaused() const override { return false; }
    };

    struct WindowResizeEvent : Event
    {
        int oWidth, oHeight;
        int width, height;

        WindowResizeEvent(int originalWidth, int originalHeight, int newWidth, int newHeight)
            : oWidth(originalWidth), oHeight(originalHeight), width(newWidth), height(newHeight) {}
        virtual EventType const getType() const override { return EventType::windowResize; }
    virtual bool            ignoreIfPaused() const override { return true; }
};

struct MouseClickedEvent : Event
{
    MouseButton button;
    Vec2f       pos;

    MouseClickedEvent(MouseButton button, Vec2f pos)
        : button(button), pos(pos) {}
    virtual EventType const getType() const override { return EventType::mouseClicked; }
    virtual bool            ignoreIfPaused() const override { return false; }
};

struct MazeMovedEvent : Event
{
    float changeX, changeY;

    MazeMovedEvent(float changeX, float changeY)
        : changeX(changeX), changeY(changeY) {}
    virtual EventType const getType() const override { return EventType::mazeMovedEvent; }
    virtual bool            ignoreIfPaused() const override { return true; }
};

struct ShowAltTileEvent : Event
{
    bool showAlt;

    ShowAltTileEvent(bool showAlt)
        : showAlt(showAlt) {}
    virtual EventType const getType() const override { return EventType::showAltTileEvent; }
    virtual bool            ignoreIfPaused() const override { return true; }
};

struct ItemTransfer : Event
{
    uint16_t    index;
    IContainer *container;

    ItemTransfer(uint16_t index, IContainer *container)
        : index(index), container(container) {}
    virtual EventType const getType() const override { return EventType::itemTransfer; }
    virtual bool            ignoreIfPaused() const override { return false; }
};

struct ChangeGUIActiveLayer : Event
{
    InGameGUILayer layer;

    ChangeGUIActiveLayer(InGameGUILayer layer)
        : layer(layer)
    {
    }
    virtual EventType const getType() const override { return EventType::changeGUILayer; }
    virtual bool            ignoreIfPaused() const override { return false; }
};

struct ChestOpenedEvent : Event
{
    IContainer *container;
    int *              activeItem;
    GUIInventoryIDCode id;

    ChestOpenedEvent(IContainer *container, int *activeItem, GUIInventoryIDCode id)
        : container(container), activeItem(activeItem), id(id) {}

    virtual EventType const getType() const override { return EventType::chestOpened; }
    virtual bool            ignoreIfPaused() const override { return true; }
};

struct PlayerResponse : Event
{
    enum Response
    {
        reject,
        accept
    };

    Response response;
```

```
        PlayerResponse(Response response)
            : response(response) {}

        virtual EventType const getType() const override { return EventType::playerResponse; }
        virtual bool            ignoreIfPaused() const override { return true; }
    };

    // TODO: Put this in a seperate file?
    struct MobDied : Event
    {
        const Mob *mob;

        MobDied(const Mob *mob)
            : mob(mob) {}

        virtual EventType const getType() const override { return EventType::mobDied; }
        virtual bool            ignoreIfPaused() const override { return true; }
    };

}   // namespace Application

#define GLFW_KEY_UNKNOWN -1

/* Printable keys */
#define GLFW_KEY_SPACE         32
#define GLFW_KEY_APOSTROPHE    39 /* ' */
#define GLFW_KEY_COMMA         44 /* , */
#define GLFW_KEY_MINUS         45 /* - */
#define GLFW_KEY_PERIOD        46 /* . */
#define GLFW_KEY_SLASH         47 /* / */
#define GLFW_KEY_0             48
#define GLFW_KEY_1             49
#define GLFW_KEY_2             50
#define GLFW_KEY_3             51
#define GLFW_KEY_4             52
#define GLFW_KEY_5             53
#define GLFW_KEY_6             54
#define GLFW_KEY_7             55
#define GLFW_KEY_8             56
#define GLFW_KEY_9             57
#define GLFW_KEY_SEMICOLON     59 /* ; */
#define GLFW_KEY_EQUAL         61 /* = */
#define GLFW_KEY_A             65
#define GLFW_KEY_B             66
#define GLFW_KEY_C             67
#define GLFW_KEY_D             68
#define GLFW_KEY_E             69
#define GLFW_KEY_F             70
#define GLFW_KEY_G             71
#define GLFW_KEY_H             72
#define GLFW_KEY_I             73
#define GLFW_KEY_J             74
#define GLFW_KEY_K             75
#define GLFW_KEY_L             76
#define GLFW_KEY_M             77
#define GLFW_KEY_N             78
#define GLFW_KEY_O             79
#define GLFW_KEY_P             80
#define GLFW_KEY_Q             81
#define GLFW_KEY_R             82
#define GLFW_KEY_S             83
#define GLFW_KEY_T             84
#define GLFW_KEY_U             85
#define GLFW_KEY_V             86
#define GLFW_KEY_W             87
#define GLFW_KEY_X             88
#define GLFW_KEY_Y             89
#define GLFW_KEY_Z             90
#define GLFW_KEY_LEFT_BRACKET  91  /* [ */
#define GLFW_KEY_BACKSLASH     92  /* \ */
#define GLFW_KEY_RIGHT_BRACKET 93  /* ] */
#define GLFW_KEY_GRAVE_ACCENT  96  /* ` */
#define GLFW_KEY_WORLD_1       161 /* non-US #1 */
#define GLFW_KEY_WORLD_2       162 /* non-US #2 */

/* Function keys */
#define GLFW_KEY_ESCAPE        256
#define GLFW_KEY_ENTER         257
#define GLFW_KEY_TAB           258
#define GLFW_KEY_BACKSPACE     259
```

```
#define GLFW_KEY_INSERT         260
#define GLFW_KEY_DELETE         261
#define GLFW_KEY_RIGHT          262
#define GLFW_KEY_LEFT           263
#define GLFW_KEY_DOWN           264
#define GLFW_KEY_UP             265
#define GLFW_KEY_PAGE_UP        266
#define GLFW_KEY_PAGE_DOWN      267
#define GLFW_KEY_HOME           268
#define GLFW_KEY_END            269
#define GLFW_KEY_CAPS_LOCK      280
#define GLFW_KEY_SCROLL_LOCK    281
#define GLFW_KEY_NUM_LOCK       282
#define GLFW_KEY_PRINT_SCREEN   283
#define GLFW_KEY_PAUSE          284
#define GLFW_KEY_F1             290
#define GLFW_KEY_F2             291
#define GLFW_KEY_F3             292
#define GLFW_KEY_F4             293
#define GLFW_KEY_F5             294
#define GLFW_KEY_F6             295
#define GLFW_KEY_F7             296
#define GLFW_KEY_F8             297
#define GLFW_KEY_F9             298
#define GLFW_KEY_F10            299
#define GLFW_KEY_F11            300
#define GLFW_KEY_F12            301
#define GLFW_KEY_F13            302
#define GLFW_KEY_F14            303
#define GLFW_KEY_F15            304
#define GLFW_KEY_F16            305
#define GLFW_KEY_F17            306
#define GLFW_KEY_F18            307
#define GLFW_KEY_F19            308
#define GLFW_KEY_F20            309
#define GLFW_KEY_F21            310
#define GLFW_KEY_F22            311
#define GLFW_KEY_F23            312
#define GLFW_KEY_F24            313
#define GLFW_KEY_F25            314
#define GLFW_KEY_KP_0           320
#define GLFW_KEY_KP_1           321
#define GLFW_KEY_KP_2           322
#define GLFW_KEY_KP_3           323
#define GLFW_KEY_KP_4           324
#define GLFW_KEY_KP_5           325
#define GLFW_KEY_KP_6           326
#define GLFW_KEY_KP_7           327
#define GLFW_KEY_KP_8           328
#define GLFW_KEY_KP_9           329
#define GLFW_KEY_KP_DECIMAL     330
#define GLFW_KEY_KP_DIVIDE      331
#define GLFW_KEY_KP_MULTIPLY    332
#define GLFW_KEY_KP_SUBTRACT    333
#define GLFW_KEY_KP_ADD         334
#define GLFW_KEY_KP_ENTER       335
#define GLFW_KEY_KP_EQUAL       336
#define GLFW_KEY_LEFT_SHIFT     340
#define GLFW_KEY_LEFT_CONTROL   341
#define GLFW_KEY_LEFT_ALT       342
#define GLFW_KEY_LEFT_SUPER     343
#define GLFW_KEY_RIGHT_SHIFT    344
#define GLFW_KEY_RIGHT_CONTROL  345
#define GLFW_KEY_RIGHT_ALT      346
#define GLFW_KEY_RIGHT_SUPER    347
#define GLFW_KEY_MENU           348

#define GLFW_KEY_LAST GLFW_KEY_MENU

#define GLFW_MOD_SHIFT 0x0001

#define GLFW_MOD_CONTROL 0x0002

#define GLFW_MOD_ALT 0x0004

/* Mouse buttons */
#define GLFW_MOUSE_BUTTON_1       0
#define GLFW_MOUSE_BUTTON_2       1
#define GLFW_MOUSE_BUTTON_3       2
#define GLFW_MOUSE_BUTTON_4       3
```

```
#define GLFW_MOUSE_BUTTON_5      4
#define GLFW_MOUSE_BUTTON_6      5
#define GLFW_MOUSE_BUTTON_7      6
#define GLFW_MOUSE_BUTTON_8      7
#define GLFW_MOUSE_BUTTON_LAST   GLFW_MOUSE_BUTTON_8
#define GLFW_MOUSE_BUTTON_LEFT   GLFW_MOUSE_BUTTON_1
#define GLFW_MOUSE_BUTTON_RIGHT  GLFW_MOUSE_BUTTON_2
#define GLFW_MOUSE_BUTTON_MIDDLE GLFW_MOUSE_BUTTON_3
```

../src/utils/Event.h

utils/Event.cpp

```
#include "Event.h"

#include "Application.h"

#include "Core.h"
#include "Log.h"
#include "ShaderEffectsManager.h"

namespace Event
{
    static uint8_t mouseClickedDelay;

    static void key_callback(GLFWwindow *window, int key, int scancode, int action, int mods)
    {
        KeyboardEvent e(key, scancode, action, mods);
        Application::callEvent(e, true);
    }

    static void window_size_callback(GLFWwindow *window, int width, int height)
    {
        //windowSizeChange(windowWidth - width, windowHeight - height);
        Application::updateWindowSize(width, height);
        glViewport(0, 0, width, height);

        WindowResizeEvent e(Application::getWidth(), Application::getHeight(), width, height);
        Application::callEvent(e, true);
    }

    static void scroll_callback(GLFWwindow *window, double xoffset, double yoffset)
    {
        ScrollEvent e(xoffset, yoffset);
        Application::callEvent(e, true);
    }

    static void error_callback(int error, const char *description)
    {
        Log::error(description, LOGINFO);
    }

    static void mouse_button_callback(GLFWwindow *window, int button, int action, int mods)
    {
        if(mouseClickedDelay == 0)
        {   // TODO: Make this so it knows the action and mods and get rid of the mouse delay
            MouseButton        mButton = static_cast<MouseButton>(button);
            MouseClickedEvent e(mButton, getMousePos());
            Application::callEvent(e, true);
            mouseClickedDelay = 10;
        }
    }

    void update()
    {
        if(mouseClickedDelay > 0)
            mouseClickedDelay--;
    }

    void init()
    {
        GLFWwindow *window = static_cast<GLFWwindow *>(Application::getWindow());
        glfwSetKeyCallback(window, key_callback);    // TODO: Change this to one function
        glfwSetWindowSizeCallback(window, window_size_callback);
        glfwSetScrollCallback(window, scroll_callback);
        glfwSetErrorCallback(error_callback);
        glfwSetMouseButtonCallback(window, mouse_button_callback);

        mouseClickedDelay = 0;
    }
    bool isKeyPressed(int key)
    {
        int keystate = glfwGetKey(static_cast<GLFWwindow *>(Application::getWindow()), key);
        return keystate == GLFW_PRESS || keystate == GLFW_REPEAT;
    }

    Vec2f getMousePos()
    {
        double xPos, yPos;
        glfwGetCursorPos((GLFWwindow *) Application::getWindow(), &xPos, &yPos);

        return {(float) xPos, Application::getHeight() - (float) yPos};
    }

}   // namespace Application
```

../src/utils/Event.cpp

3.2.5 Log

utils/Log.h

```
#pragma once

#include <sstream>
#include <string>

#define LOGINFO __FILE__, __LINE__   // This is for just quickly getting information needed if there is an erro

enum LogColour
{
    reset = 0,
    critical,
    error,
    warning,
    info,
    debug,
    variable
};

class Log
{
  private:
    Log();

    static bool       outputting;
    std::string       logFile         = "Logs/Default.log";
    const std::string criticalMessage = "CRITICAL";
    const std::string errorMessage    = "ERROR";
    const std::string warningMessage  = "WARNING";
    const std::string debugMessage    = "DEBUG";
    const std::string variableMessage = "VAR";
    const std::string defaultMessage  = "INFO";

    void output(const std::string &type, const char *message, const char *filepath, int line, LogColour colour);
    void setConsoleColour(LogColour c);

    void criticalImpl(const char *message, const char *file, int line);
    void errorImpl(const char *message, const char *file, int line);
    void warningImpl(const char *message);
    void infoImpl(const char *message);
    void debugImpl(const char *message);

    template <typename T>
    void variableImpl(const char *name, T var)
    {
        std::stringstream ss;
        ss << name << ": " << var;
        output(variableMessage, ss.str().c_str(), "", -1, LogColour::variable);
    }

  public:
    Log(const Log &) = delete;

    static void critical(const char *message, const char *file, int line) { get().criticalImpl(message, file, line
    ); }
    static void error(const char *message, const char *file, int line) { get().errorImpl(message, file, line); }
    static void warning(const char *message) { get().warningImpl(message); }
    static void info(const char *message) { get().infoImpl(message); }
    static void debug(const char *message) { get().debugImpl(message); }
    template <typename T>
    static void variable(const char *name, T var)
    {
        get().variableImpl(name, var);
    }

    static Log &get()
    {
        static Log instance;
        return instance;
    }
};
```

../src/utils/Log.h

utils/Log.cpp

```
#include "Log.h"


#ifdef IS_ON_WINDOWS
    #include <windows.h>
    #include <conio.h>
//COLORS LIST
//1: Blue
//2: Green
//3: Cyan
//4: Red
//5: Purple
//6: Yellow (Dark)
//7: Default white
//8: Gray/Grey
//9: Bright blue
//10: Bright green
//11: Bright cyan
//12: Bright red
//13: Pink/Magenta
//14: Yellow
//15: Bright white
#else
    #define RESET       "\033[0m"
    #define BLACK       "\033[30m"          /* Black */
    #define RED         "\033[31m"          /* Red */
    #define GREEN       "\033[32m"          /* Green */
    #define YELLOW      "\033[33m"          /* Yellow */
    #define BLUE        "\033[34m"          /* Blue */
    #define MAGENTA     "\033[35m"          /* Magenta */
    #define CYAN        "\033[36m"          /* Cyan */
    #define WHITE       "\033[37m"          /* White */
    #define BOLDBLACK   "\033[1m\033[30m"   /* Bold Black */
    #define BOLDRED     "\033[1m\033[31m"   /* Bold Red */
    #define BOLDGREEN   "\033[1m\033[32m"   /* Bold Green */
    #define BOLDYELLOW  "\033[1m\033[33m"   /* Bold Yellow */
    #define BOLDBLUE    "\033[1m\033[34m"   /* Bold Blue */
    #define BOLDMAGENTA "\033[1m\033[35m"   /* Bold Magenta */
    #define BOLDCYAN    "\033[1m\033[36m"   /* Bold Cyan */
    #define BOLDWHITE   "\033[1m\033[37m"   /* Bold White */
#endif

#include "LogHeaders.h"

bool Log::outputting = false;

Log::Log()
{
    time_t      rawtime;
    struct tm *timeinfo;
    char        buffer[80];

    time(&rawtime);
    timeinfo = localtime(&rawtime);

    strftime(buffer, sizeof(buffer), "%d-%m-%Y %H-%M-%S", timeinfo);
    std::string currentTime(buffer);

    if(!std::filesystem::exists("logs"))
    {
        std::cout << "Logs directory doesn't exist... creating one\n";
        std::filesystem::create_directory("logs");
        if(std::filesystem::exists("logs"))
```

```
            std::cout << "Created directory\n";
    }
    logFile = "logs/" + currentTime + ".log";

    variableImpl("Initialised logging system", logFile);
}

#ifdef IS_ON_WINDOWS
void Log::setConsoleColour(LogColour c)
{
    WORD colour = 7;
    switch(c) {
        case LogColour::critical:
            colour = FOREGROUND_RED | FOREGROUND_INTENSITY;
            break;
        case LogColour::error:
            colour = FOREGROUND_RED;
            break;
        case LogColour::warning:
            colour = 14;
            break;
        case LogColour::debug:
            colour = FOREGROUND_GREEN;
            break;
        case LogColour::variable:
            colour = FOREGROUND_INTENSITY | 13;
            break;
        default:
            colour = 7;
            break;
    }
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), colour);
}
#else
void Log::setConsoleColour(LogColour c)
{
    switch(c) {
    case LogColour::critical:
        std::cout << BOLDRED;
        break;
    case LogColour::error:
        std::cout << RED;
        break;
    case LogColour::warning:
        std::cout << YELLOW;
        break;
    case LogColour::info:
        break;
    case LogColour::debug:
        std::cout << GREEN;
        break;
    case LogColour::variable:
        std::cout << BOLDMAGENTA;
        break;
    default:
        std::cout << RESET;
        break;
        }
}
#endif

void Log::output(const std::string &type, const char *message, const char *filepath, int line, LogColour colour)
{
    // This is to stop conflicts of two threads trying to access the logging system at the same time
    while(outputting) { std::this_thread::sleep_for(std::chrono::milliseconds(100)); };
    outputting = true;
#ifdef DEBUG
    setConsoleColour(colour);
    std::cout << "[" << type << "] " << message;
    if(line != -1)
        std::cout << " " << filepath << ":" << line;
    setConsoleColour(LogColour::reset);
    std::cout << std::endl;
#endif
    std::ofstream file(logFile, std::ios_base::app);   // The std::ios_base::app allows it to write at end
    file << "[" << type << "] " << message;
    if(line != -1)
        file << " " << filepath << ":" << line;
    file << std::endl;
    file.close();
```

```cpp
    outputting = false;
}

void Log::criticalImpl(const char *message, const char *file, int line)
{
    output(criticalMessage, message, file, line, LogColour::critical);
#ifdef DEBUG
    #ifdef IS_ON_WINDOWS
    __debugbreak();
    #else
    __builtin_trap();
    #endif
#endif
}

void Log::errorImpl(const char *message, const char *file, int line)
{
    output(errorMessage, message, file, line, LogColour::error);
}

void Log::warningImpl(const char *message)
{
    output(warningMessage, message, "", -1, LogColour::warning);
}

void Log::infoImpl(const char *message)
{
    output(defaultMessage, message, "", -1, LogColour::info);
}

void Log::debugImpl(const char *message)
{
    output(debugMessage, message, "", -1, LogColour::debug);
}
```

../src/utils/Log.cpp

3.2.6 Random Generation

utils/RandomGen.h

```cpp
#pragma once

#include <random>
#include <vector>

#include "Item.h"

class Random
{
  private:
    std::default_random_engine generator;

    int getNumImpl(int min, int max);
    int getWeightedNumImpl(std::vector<float> nums);
    Item *getItemImpl();
    int    reverseNum(int num);

    Random();

  public:
    Random(const Random &) = delete;

    static int getNum(int min, int max) { return get().getNumImpl(min, max); }
    static int getWeightedNum(std::vector<float> nums) { return get().getWeightedNumImpl(nums); }

    static Item *getItem() { return get().getItemImpl(); }

    static Random &get()
    {
        static Random instance;
        return instance;
    }
};
```

../src/utils/RandomGen.h

utils/RandomGen.cpp

```cpp
#include "RandomGen.h"
```

```
#include <time.h>

#include "Log.h"

#include "Boomerang.h"
#include "Bow.h"
#include "Crossbow.h"
#include "DarkStaff.h"
#include "EarthStaff.h"
#include "FireStaff.h"
#include "FrostStaff.h"
#include "GoldStaff.h"
#include "Potion.h"
#include "Sling.h"

Random::Random()
{
    generator.seed(reverseNum(time(NULL)));
    Log::info("Initialised random number engine");
}

int Random::getNumImpl(int min, int max)
{
    if(min == max || min > max)
    {
        Log::warning("Min and Max are the same or max is less than min!");
        return max;
    }
    std::uniform_int_distribution<int> distribution(min, max);
    return distribution(generator);
}

int Random::getWeightedNumImpl(std::vector<float> nums)
{
    std::discrete_distribution<int> distribution(nums.begin(), nums.end());
    return distribution(generator);
}

Item *Random::getItemImpl()
{
    Item *item;
    int   r = getNum(0, 2);
    if(r == 0)
    {
        int r1 = getNum(0, 100);
        if(r1 < 15)
            item = new Boomerang();
        else if(r1 < 30)
            item = new Bow();
        else if(r1 < 40)
            item = new Crossbow();
        else if(r1 < 45)
            item = new DarkStaff();
        else if(r1 < 55)
            item = new EarthStaff();
        else if(r1 < 70)
            item = new FireStaff();
        else if(r1 < 80)
            item = new FrostStaff();
        else if(r1 < 83)
            item = new GoldStaff();
        else
            item = new Sling();
    }
    else
    {
        int          r1 = getNum(0, 8);
        Potion::Type type;
        if(r1 < 2)
        {
            int temp = r1 * POTION_SPRITES;
            int r2   = getNum(0, 9);
            if(r2 > 8)
                temp += 3;
            else if(r2 > 6)
                temp += 2;
            else if(r2 > 3)
                temp += 1;
            type = static_cast<Potion::Type>(temp);
        }
        else if(r1 < 4)
        {
            int r2 = getNum(0, 2);
            if(r2 == 0)
                type = Potion::Type::MagicBook;
            else
                type = Potion::Type::Book;
        }
        else
            type = Potion::Type::Food;

        item = new Potion(type);
    }

    return item;
}

int Random::reverseNum(int num)   // This returns a number in reverse
{
    int reverse = 0, rem;
    while(num != 0)
    {
        rem     = num % 10;
        reverse = reverse * 10 + rem;
        num /= 10;
    }
    return reverse;
}
```

../src/utils/RandomGen.cpp

3.3 Rendering

3.4 Renderer

rendering/Buffer.h

```
#pragma once

#include "Log.h"

#include <vector>

template <typename T>
class Buffer : public std::vector<T *>
{
  private:
    std::array<uint32_t, 11> layersLoc;   // last layer is there for safety reasons and just stores the size of
    the vector

  public:
    Buffer()
    {
        memset(&layersLoc, 0, sizeof(layersLoc));
    }
    ~Buffer()
    {
        for(T *obj : *this)
            delete obj;
    }

    void addElement(T *obj, uint8_t layer, bool orderByYAxis)
    {
        if(layer >= layersLoc.size() - 1)   // NOTE: this skips the last layer because it for safety
        {
            Log::warning("Buffer layer given is to high!");
            layer = layersLoc.size() - 2;
        }

        int layerSize = layersLoc[layer + 1] - layersLoc[layer];

        if(orderByYAxis && layerSize > 0)
        {
            float objY = obj->position.y;
            // If the size is one it does a quick evaluation to see where to place it
            if(layerSize == 1)
            {
```

```
            auto it = std::vector<T *>::begin() + layersLoc[layer];
            if((*it)->position.y > objY)
                std::vector<T *>::insert(it + layerSize, obj);
            else
                std::vector<T *>::insert(it, obj);
        }
        else
        {
            // This is a modified binomial search to find the range of where to place the new element
            // These variables keeps track of the range of elements of the set
            int  startSub   = 0;
            int  endSub     = layerSize;
            int  index      = (startSub + endSub) / 2;
            auto layerBegin = std::vector<T *>::begin() + layersLoc[layer];

            auto getYOf = [layerBegin](int index) -> float {
                return (*(layerBegin + index))->position.y;
            };

            while(startSub != endSub && startSub < endSub)   // This continues going until the range is 0
            {
                if(index + 1 >= layerSize)
                    index = layerSize - 2;   // If it is looking at the end of the list, it minuses 1, so that
 the algorithm can work without any errors
                else if(index < 0)
                    index = 0;

                // Gets the y positions of the elements at the current index and above it
                float thisY = getYOf(index);
                float nextY = getYOf(index + 1);

                if(nextY > thisY)   // Checks to see if the vector is sorted incorrectly
                {
                    Log::critical("Buffer is not sorted correctly!", LOGINFO);
                    index = std::vector<T *>::size();
                    break;
                }

                // Checks to see if it has found a location to place the new element
                if(objY == thisY || (thisY > objY && objY > nextY))
                {
                    index++;
                    break;
                }
                else if(objY == nextY)
                {
                    index += 2;
                    break;
                }
                else if(objY < nextY)   // Otherwise it modifies the range for the binomial search to work
                    startSub = index + 2;
                else if(objY > thisY)
                    endSub = index;

                index = (startSub + endSub) / 2;   // Sets the index to be inbetween the new ranges
            }

            if(index > layerSize)
            {
                Log::warning("The index is over its range!");
                std::vector<T *>::insert(std::vector<T *>::begin() + layersLoc[layer + 1], obj);
            }
            else
            {
                if(index > 0 && index < layerSize)
                {
                    if(getYOf(index - 1) < objY || objY < getYOf(index))
                    {
                        Log::critical("Index is incorrectly calculated", LOGINFO);
                    }
                }
                std::vector<T *>::insert(layerBegin + index, obj);   // Inserts the object at the correct
position
            }
        }
    }
    else
    {
        // Inserts it at the end of the layer wanted
        if(layersLoc[layer + 1] == std::vector<T *>::size())
```

```
                std::vector<T *>::push_back(obj);
            else
                std::vector<T *>::insert(std::vector<T *>::begin() + layersLoc[layer + 1], obj);
        }

        // updates locations of the layers
        for(int i = layer + 1; i < layersLoc.size(); i++)
            layersLoc[i]++;
    }

    uint32_t getLayerPos(uint8_t layer)
    {
        if(layer >= layersLoc.size())   // NOTE: this skips the last layer because it for safety
        {
            Log::warning("Buffer layer given is to high!");
            layer = layersLoc.size() - 1;
        }
        return layersLoc[layer];
    }
    uint32_t getLayerSize(uint8_t layer)
    {
        if(layer >= layersLoc.size())   // NOTE: this skips the last layer because it for safety
        {
            Log::warning("Buffer layer given is to high!");
            layer = layersLoc.size() - 1;
        }
        return layersLoc[layer + 1] - layersLoc[layer];
    }

    void clear()
    {
        // Clears the layers information
        memset(&layersLoc, 0, sizeof(layersLoc));
        std::vector<T *>::clear();
    }
};
```

../src/rendering/Buffer.h

rendering/RenderVertex.h

```
#pragma once

#include <GLM.h>

// This is all the vertices, set in the correct layout to be directly copied to the vertex buffer without any alterations
// Each type of vertex is for a different shader
struct Vertex
{
    glm::vec2 position;

    Vertex() {}
    Vertex(glm::vec2 position)
        : position(position) {}
};

struct TexturedVertex : public Vertex   // For the sprite shader
{
    glm::vec2 texCoords;
    float     texID;

    TexturedVertex() {}
    TexturedVertex(glm::vec2 position, glm::vec2 texCoords, float texID)
        : Vertex(position), texCoords(texCoords), texID(texID) {}
};

struct ColouredVertex : public Vertex   // For the simple shader
{
    glm::vec4 colour;

    ColouredVertex() {}
    ColouredVertex(glm::vec2 position, glm::vec4 colour)
        : Vertex(position), colour(colour) {}
};

struct TextVertex : public TexturedVertex   // For the text shader
{
    glm::vec4 colour;

    TextVertex() {}
    TextVertex(glm::vec2 position, glm::vec2 texCoords, float texID, glm::vec4 colour)
        : TexturedVertex(position, texCoords, texID), colour(colour) {}
};
```

../src/rendering/RenderVertex.h

rendering/RenderObject.h

```
#pragma once

#include <GLM.h>
#include <array>

#include "RenderVertex.h"
#include "Sprite.h"
#include "Texture.h"

// Text rendering objects
struct Character
{
    Texture *  texture;
    glm::ivec2 size;      // Size of glyph
    glm::ivec2 bearing;   // Offset from baseline to left/top of glyph
    uint16_t   advance;   // Offset to advance to next glyph

    ~Character()
    {
        if(texture)
            delete texture;
    }
};

// These are the objects that are stored in the buffer, and store all the information needed so they can be
    converted and transferred to the vertex buffer
// This is most definitely less efficient then just converting them and storing them in the buffer
// However, this allows me to easily order and manipulate them while stored in the buffer,
// so I believe the trade off is worth it

struct RenderObject
{
    glm::vec2 position;
    float     width, height;
    double    rotation;
    bool      centered;

    RenderObject() {}
    RenderObject(glm::vec2 position, float width, float height, double rotation, bool centered)
        : position(position), width(width), height(height), rotation(rotation), centered(centered) {}

    // Gets the size of each array of vertices returned by 'convertToVertices'
    virtual uint32_t      getSizeOfVertices() { return 4 * sizeof(Vertex); }
    std::array<Vertex, 4> convertToVertices();
};

struct ColouredObject : public RenderObject
{
    glm::vec4 colour;

    ColouredObject() {}
    ColouredObject(glm::vec2 position, float width, float height, double rotation, bool centered, glm::vec4 colour
    )
        : RenderObject(position, width, height, rotation, centered), colour(colour) {}

    // Gets the size of each array of vertices returned by 'convertToColouredVertices'
    virtual uint32_t              getSizeOfVertices() override { return 4 * sizeof(ColouredVertex); }
    std::array<ColouredVertex, 4> convertToColouredVertices();
};

struct TexturedObject : public RenderObject
{
    Sprite::ID spriteID;

    TexturedObject() {}
    TexturedObject(glm::vec2 position, float width, float height, double rotation, bool centered, Sprite::ID
    spriteID)
        : RenderObject(position, width, height, rotation, centered), spriteID(spriteID) {}

    // Gets the size of each array of vertices returned by 'convertToTexturedVertices'
    virtual uint32_t              getSizeOfVertices() override { return 4 * sizeof(TexturedVertex); }
    std::array<TexturedVertex, 4> convertToTexturedVertices(uint8_t texSlot);
};
```

```cpp
struct TextObject : public ColouredObject
{
    std::string text;
    float       scale;

    TextObject(std::string text, float scale, glm::vec2 position, float width, float height, double rotation, glm
    ::vec4 colour, bool centered)
        : ColouredObject(position, width, height, rotation, centered, colour), text(text), scale(scale) {}

    // Gets the size of each array of vertices returned by 'convertToCharacterVertices'
    virtual uint32_t          getSizeOfVertices() override { return 4 * sizeof(TextVertex); }
    // This function is slightly different from the rest, however this is because it stores a string and so is
    rendered
    // by going through each character, so this takes in the character and the offset when creating the vertices
    // This allows to correctly render rotated strings
    std::array<TextVertex, 4> convertCharacterToVertices(Character *ch, float xOffset, uint8_t texSlot);
};
```

../src/rendering/RenderObject.h

rendering/RenderObject.cpp

```cpp
#include "RenderObject.h"

std::array<Vertex, 4> RenderObject::convertToVertices()
{
    // Creates a 2d rotation matrix, so that the object can be rotated
    glm::mat2 rotationMatrix({glm::cos(rotation), -glm::sin(rotation)}, {glm::sin(rotation), glm::cos(rotation)});

    // Gets the point at which to rotate around
    float leftPoint, rightPoint, topPoint, bottomPoint;
    if(centered)
    {
        // This centers the object
        float xHalfSize = width / 2;
        float yHalfSize = height / 2;
        leftPoint       = -xHalfSize;
        rightPoint      = xHalfSize;
        topPoint        = yHalfSize;
        bottomPoint     = -yHalfSize;
    }
    else
    {
        leftPoint   = 0;
        rightPoint  = width;
        topPoint    = height;
        bottomPoint = 0;
    }

    // Creates 4 vertices that create the square
    Vertex v0(rotationMatrix * glm::vec2(leftPoint, bottomPoint) + position);
    Vertex v1(rotationMatrix * glm::vec2(rightPoint, bottomPoint) + position);
    Vertex v2(rotationMatrix * glm::vec2(rightPoint, topPoint) + position);
    Vertex v3(rotationMatrix * glm::vec2(leftPoint, topPoint) + position);

    return {v0, v1, v2, v3};
}

std::array<ColouredVertex, 4> ColouredObject::convertToColouredVertices()
{
    // Creates a 2d rotation matrix, so that the object can be rotated
    glm::mat2 rotationMatrix({glm::cos(rotation), -glm::sin(rotation)}, {glm::sin(rotation), glm::cos(rotation)});

    // Gets the point at which to rotate around
    float leftPoint, rightPoint, topPoint, bottomPoint;
    if(centered)
    {
        // This centers the object
        float xHalfSize = width / 2;
        float yHalfSize = height / 2;

        leftPoint   = -xHalfSize;
        rightPoint  = xHalfSize;
        topPoint    = yHalfSize;
        bottomPoint = -yHalfSize;
    }
    else
    {
        leftPoint   = 0;
        rightPoint  = width;
        topPoint    = height;
        bottomPoint = 0;
    }

    // Creates 4 vertices that create the rectangle
    ColouredVertex v0(
        rotationMatrix * glm::vec2(leftPoint, bottomPoint) + position,
        colour);
    ColouredVertex v1(
        rotationMatrix * glm::vec2(rightPoint, bottomPoint) + position,
        colour);
    ColouredVertex v2(
        rotationMatrix * glm::vec2(rightPoint, topPoint) + position,
        colour);
    ColouredVertex v3(
        rotationMatrix * glm::vec2(leftPoint, topPoint) + position,
        colour);

    return {v0, v1, v2, v3};
}

std::array<TexturedVertex, 4> TexturedObject::convertToTexturedVertices(uint8_t texSlot)
{
    // Creates a 2d rotation matrix, so that the object can be rotated
    glm::mat2 rotationMatrix({glm::cos(rotation), -glm::sin(rotation)}, {glm::sin(rotation), glm::cos(rotation)});

    // Gets the point at which to rotate around
    float leftPoint, rightPoint, topPoint, bottomPoint;
    if(centered)
    {
        // This centers the object
        float xHalfSize = width / 2;
        float yHalfSize = height / 2;
        leftPoint       = -xHalfSize;
        rightPoint      = xHalfSize;
        topPoint        = yHalfSize;
        bottomPoint     = -yHalfSize;
    }
    else
    {
        leftPoint   = 0;
        rightPoint  = width;
        topPoint    = height;
        bottomPoint = 0;
    }

    // Creates 4 vertices that create the sprite
    TexturedVertex v0(
        rotationMatrix * glm::vec2(leftPoint, bottomPoint) + position,
        {0.0f, 0.0f},
        texSlot);

    TexturedVertex v1(
        rotationMatrix * glm::vec2(rightPoint, bottomPoint) + position,
        {1.0f, 0.0f},
        texSlot);

    TexturedVertex v2(
        rotationMatrix * glm::vec2(rightPoint, topPoint) + position,
        {1.0f, 1.0f},
        texSlot);

    TexturedVertex v3(
        rotationMatrix * glm::vec2(leftPoint, topPoint) + position,
        {0.0f, 1.0f},
        texSlot);

    return {v0, v1, v2, v3};
}

std::array<TextVertex, 4> TextObject::convertCharacterToVertices(Character *ch, float xOffset, uint8_t texSlot)
{
    float newScale = scale / 100;

    float xPos = position.x + ch->bearing.x * newScale + xOffset;
    float yPos = position.y - (ch->size.y - ch->bearing.y) * newScale;

    float w = ch->size.x * newScale;
    float h = ch->size.y * newScale;
```

```cpp
    // Creates a 2d rotation matrix, so that the object can be rotated
    glm::mat2 rotationMatrix({glm::cos(rotation), -glm::sin(rotation)}, {glm::sin(rotation), glm::cos(rotation)});

    // Gets the point at which to rotate around
    float leftPoint, rightPoint, topPoint, bottomPoint;
    if(centered)
    {
        // This centers the object
        float xHalfSize = width / 2;
        float yHalfSize = height / 2;
        leftPoint       = -xHalfSize;
        rightPoint      = -xHalfSize + w;
        topPoint        = -yHalfSize + h;
        bottomPoint     = -yHalfSize;
    }
    else
    {
        leftPoint   = 0;
        rightPoint  = w;
        topPoint    = h;
        bottomPoint = 0;
    }

    // Creates 4 vertices that create the character
    TextVertex v0(
        rotationMatrix * glm::vec2(leftPoint, topPoint) + glm::vec2(xPos, yPos),
        {0.0f, 0.0f},
        texSlot,
        colour);
    TextVertex v1(
        rotationMatrix * glm::vec2(rightPoint, topPoint) + glm::vec2(xPos, yPos),
        {1.0f, 0.0f},
        texSlot,
        colour);
    TextVertex v2(
        rotationMatrix * glm::vec2(rightPoint, bottomPoint) + glm::vec2(xPos, yPos),
        {1.0f, 1.0f},
        texSlot,
        colour);
    TextVertex v3(
        rotationMatrix * glm::vec2(leftPoint, bottomPoint) + glm::vec2(xPos, yPos),
        {0.0f, 1.0f},
        texSlot,
        colour);

    return {v0, v1, v2, v3};
}
```

../src/rendering/RenderObject.cpp

rendering/Renderer.h

```cpp
#pragma once

#include <array>
#include <memory>
#include <unordered_map>
#include <vector>

#include "RenderObject.h"
#include "RenderVertex.h"

#include "Buffer.h"
#include "IndexBuffer.h"
#include "Shader.h"
#include "ShaderEffectsManager.h"
#include "Texture.h"
#include "Utils.h"
#include "VertexArray.h"

// Render is a singleton and handles all the rendering
// all the static functions are there so you can call Render::rectangle(...) instead of Render::get().rectangle(...)

class Render
{
  public:
    Render(const Render &) = delete;
    ~Render();

    static void render(std::vector<uint16_t> &shaderEffects) { get().renderImpl(shaderEffects); }
    static void sprite(float x, float y, double rotation, float size, Sprite::ID spriteID, uint8_t layer, bool isOverlay = false)
    {
        get().spriteImpl(x, y, rotation, size, size, spriteID, layer, isOverlay);
    }
    static void sprite(float x, float y, double rotation, float width, float height, Sprite::ID spriteID, uint8_t layer, bool isOverlay = false)
    {
        get().spriteImpl(x, y, rotation, width, height, spriteID, layer, isOverlay);
    }
    static void text(std::string &text, float x, float y, float scale, glm::vec4 colour, uint8_t layer, bool isCentered = false, bool isOverlay = false)
    {
        get().textImpl(text, x, y, scale, colour, layer, isCentered, isOverlay);
    }
    static void hoverText(std::string &text, float x, float y, float scale, glm::vec4 textColour, glm::vec4 backgroundColour, uint8_t layer, bool isOverlay = false)
    {
        get().hoverTextImpl(text, x, y, scale, textColour, backgroundColour, layer, isOverlay);
    }
    static void rectangle(float x, float y, double rotation, float width, float height, glm::vec4 colour, uint8_t layer, bool isCentered = true, bool isOverlay = false)
    {
        get().rectangleImpl(x, y, rotation, width, height, colour, layer, isCentered, isOverlay);
    }
    static void rectangle(float x, float y, float width, float height, glm::vec4 colour, float borderWidth, glm::vec4 borderColour, uint8_t layer, bool isCentered = true, bool isOverlay = false)
    {
        get().rectangleImpl(x, y, width, height, colour, borderWidth, borderColour, layer, isCentered, isOverlay);
    }

    static float        getTextWidth(std::string &text, float scale) { return get().getTextWidthImpl(text, scale); }
    static float        getTextHeight(std::string &text, float scale) { return get().getTextHeightImpl(text, scale); }
    static CollisionBox getTextCollisionBox(std::string &text, float scale) { return get().getTextCollisionBoxImpl(text, scale); }

    static void orderBuffersByYAxis() { get().orderBuffersByYAxisImpl(); }

    static Render &get()
    {
        static Render instance;
        return instance;
    }

  private:
    static Render s_Instance;

    // The index and vertex buffers are shared accross the vertex arrays, so there is only need for one of each
    std::unique_ptr<IndexBuffer>  m_IndexBuffer;
    std::unique_ptr<VertexBuffer> m_VertexBuffer;

    // Each vertex array are for each shader used, and are used so I don't have to keep reapplying the vertex buffer layout
    std::unique_ptr<VertexArray> m_TextVAO;
    std::unique_ptr<VertexArray> m_SpriteVAO;
    std::unique_ptr<VertexArray> m_SimpleVAO;

    // The shaders used when rendering
    std::unique_ptr<Shader> m_TextShader;
    std::unique_ptr<Shader> m_SpriteShader;
    std::unique_ptr<Shader> m_SimpleShader;

    std::unordered_map<char, Character> characters;        // This stores all the information for each character of text needed when rendering

    Buffer<TextObject> m_TextObjBuffer;   // This acts as a buffer for the text, so that it can render it in one go

    // These are the buffers that store the simple coloured rectanges information, so that they can be rendered properly later on
    Buffer<ColouredObject> m_ObjectBuffer;

    // This is the buffer for all the sprites
    Buffer<TexturedObject> m_SpriteBuffer;

    // Settings
    bool orderBuffersByYAxisSetting;   // This will order the sprite buffer so that they are rendered in the correct order
    Render();

    // Rendering functions
    void renderImpl(std::vector<uint16_t> &shaderEffects);
    void simpleRender(uint8_t startLayer, uint8_t endLayer);
    void spriteRender(uint8_t startLayer, uint8_t endLayer);
    void textRender(uint8_t startLayer, uint8_t endLayer);
    void draw(VertexArray &vao) const;

    // Functions for adding objects to buffers
    void spriteImpl(float x, float y, double rotation, float width, float height, Sprite::ID spriteID, uint8_t
    layer, bool isOverlay);
    void textImpl(std::string &text, float x, float y, float scale, glm::vec4 colour, uint8_t layer, bool
    isCentered, bool isOverlay);   // TODO: Make this order better
    void hoverTextImpl(std::string &inpText, float x, float y, float scale, glm::vec4 textColour, glm::vec4
    backgroundColour, uint8_t layer, bool isOverlay);
    void rectangleImpl(float x, float y, double rotation, float width, float height, glm::vec4 colour, uint8_t
    layer, bool isCentered, bool isOverlay);
    void rectangleImpl(float x, float y, float width, float height, glm::vec4 colour, float borderWidth, glm::vec4
     borderColour, uint8_t layer, bool isCentered, bool isOverlay);

    // Getters
    float        getTextWidthImpl(std::string &text, float scale);
    float        getTextHeightImpl(std::string &text, float scale);
    CollisionBox getTextCollisionBoxImpl(std::string &text, float scale);

    // Settings
    void orderBuffersByYAxisImpl();
};
```

../src/rendering/Renderer.h

rendering/Renderer.cpp

```
#include "Renderer.h"

#include "Application.h"
#include "ShaderEffectsManager.h"
#include "Sprite.h"
#include "VertexBufferLayout.h"

#include <ft2build.h>
#include FT_FREETYPE_H

Render::Render()
    : orderBuffersByYAxisSetting(false)
{
    // Text initialisation
    FT_Library ft;
    if(FT_Init_FreeType(&ft))
        Log::critical("FREETYPE: Could not init FreeType", LOGINFO);

    FT_Face face;
    if(FT_New_Face(ft, "res/fonts/FiraCode.ttf", 0, &face))
        Log::critical("FREETYPE: Failed to load font", LOGINFO);

    FT_Set_Pixel_Sizes(face, 0, 48);
    if(FT_Load_Char(face, 'X', FT_LOAD_RENDER))
        Log::critical("FREETYPE: Failed to load Glyph", LOGINFO);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);   // disable byte-alignment restriction

    float maxHeight = 0;
    float minHeight = 0;
    for(unsigned char c = 0; c < 128; c++)
    {
        // Loads each glyph
        if(FT_Load_Char(face, c, FT_LOAD_RENDER))
        {
            std::cout << "ERROR::FREETYPE: Failed to load Glyph" << std::endl;
            continue;
        }
        // Adds the character characters map, so it can be quickly retrieved later
        Character character = {
            nullptr,
            glm::ivec2(face->glyph->bitmap.width, face->glyph->bitmap.rows),
            glm::ivec2(face->glyph->bitmap_left, face->glyph->bitmap_top),
            (uint16_t) face->glyph->advance.x,
        };
        characters.insert(std::pair<char, Character>(c, character));
        characters[c].texture = new Texture(face->glyph->bitmap.width, face->glyph->bitmap.rows, face->glyph->
bitmap.buffer);

    if(characters[c].bearing.y > maxHeight)
        maxHeight = characters[c].bearing.y;
    if(characters[c].bearing.y - characters[c].size.y < minHeight)
        minHeight = characters[c].bearing.y - characters[c].size.y;
}

Log::variable("Max Height", maxHeight - minHeight);

FT_Done_Face(face);
FT_Done_FreeType(ft);

Log::info("Text initialised");

// Shaders setup
int samplers[32];   // This is used for when rendering multiple textures with one draw function
for(int i = 0; i < 32; i++)
    samplers[i] = i;

m_SpriteShader = std::make_unique<Shader>("res/shaders/SpriteShader.glsl");
m_SpriteShader->setUniform1iv("u_Textures", 32, samplers);

m_TextShader = std::make_unique<Shader>("res/shaders/TextShader.glsl");
m_TextShader->setUniform1iv("u_Textures", 32, samplers);

m_SimpleShader = std::make_unique<Shader>("res/shaders/SimpleShader.glsl");

// Buffers setup
uint32_t maxVertices = 3528;

m_VertexBuffer = std::make_unique<VertexBuffer>(nullptr, (uint16_t) sizeof(float) * 5 * maxVertices);
m_IndexBuffer = std::make_unique<IndexBuffer>((maxVertices / 4) * 6);


// Sprite VAO set up
// Creates the VAO and links it to the vertex buffer
m_SpriteVAO = std::make_unique<VertexArray>();
m_SpriteVAO->bind();

// Creates the layout of variables being pushed to the shader
VertexBufferLayout spriteLayout = {
    {ShaderDataType::Float2, "position"},
    {ShaderDataType::Float2, "texCoord"},
    {ShaderDataType::Float, "texIndex"},
};
m_SpriteVAO->addBuffer(*m_VertexBuffer, spriteLayout);   // Adds it to the VAO
m_IndexBuffer->bind();

// Unbinds everything
m_SpriteVAO->unbind();
m_IndexBuffer->unbind();

// Text VAO set up
// Creates the VAO and links it to the vertex buffer
m_TextVAO = std::make_unique<VertexArray>();
m_TextVAO->bind();
// Creates the layout of variables being pushed to the shader
VertexBufferLayout textLayout = {
    {ShaderDataType::Float2, "position"},
    {ShaderDataType::Float2, "texCoord"},
    {ShaderDataType::Float, "texIndex"},
    {ShaderDataType::Float4, "texColour"},
};
m_TextVAO->addBuffer(*m_VertexBuffer, textLayout);   // Adds it to the VAO
m_IndexBuffer->bind();
// Unbinds everything
m_TextVAO->unbind();
m_IndexBuffer->unbind();

// Simple VAO set up
// Creates the VAO and links it to the vertex buffer
m_SimpleVAO = std::make_unique<VertexArray>();
m_SimpleVAO->bind();

// Creates the layout of variables being pushed to the shader
VertexBufferLayout simpleLayout = {
    {ShaderDataType::Float2, "position"},
    {ShaderDataType::Float4, "colour"},
};
```

```
    m_SimpleVAO->addBuffer(*m_VertexBuffer, simpleLayout);   // Adds it to the VAO
    m_IndexBuffer->bind();
    // Unbinds everything
    m_SimpleVAO->unbind();
    m_IndexBuffer->unbind();

    Log::info("Buffer and VAOs set up");

    Log::info("Renderer initialised");
}

Render::~Render()
{
    // Deletes all the objects stored as pointers
    for(TextObject *obj : m_TextObjBuffer)
        delete obj;
    for(ColouredObject *obj : m_ObjectBuffer)
        delete obj;
    for(TexturedObject *obj : m_SpriteBuffer)
        delete obj;
    Log::info("Renderer destroyed");
}

void Render::renderImpl(std::vector<uint16_t> &shaderEffects)
{
    // resets the default effects
    m_SpriteShader->setUniform4f("u_Zoom", 1.0f, 1.0f, 1.0f, 1.0f);
    m_TextShader->setUniform4f("u_Zoom", 1.0f, 1.0f, 1.0f, 1.0f);
    m_SimpleShader->setUniform4f("u_Zoom", 1.0f, 1.0f, 1.0f, 1.0f);
    m_SpriteShader->setUniformMat4f("u_MVP", Application::getProj());
    m_TextShader->setUniformMat4f("u_MVP", Application::getProj());
    m_SimpleShader->setUniformMat4f("u_MVP", Application::getProj());

    // Sets effects given
    for(uint16_t id : shaderEffects)
    {
        if(id == 0)   // Checks if the effect exists
        {
            Log::warning("Trying to use effect that doesn't exist!");
            continue;
        }
        Effect::ShaderEffect *e = Effect::ShaderEffectsManager::getShaderEffect(id);
        if(e->forSimpleShader())
            e->setEffect(*m_SimpleShader);
        if(e->forSpriteShader())
            e->setEffect(*m_SpriteShader);
        if(e->forTextShader())
            e->setEffect(*m_TextShader);
    }
    // Renders
    // TODO: Put everything into one buffer?
    enum class RenderBuffer
    {
        None,
        Simple,
        Sprite,
        Text
    };
    uint8_t      startLayer    = 0;
    RenderBuffer currentBuffer = RenderBuffer::None;

    auto renderCurrentBuffer = [this](RenderBuffer currentBuffer, uint8_t startLayer, uint8_t endLayer) {
        switch(currentBuffer)
        {
        case RenderBuffer::Simple:
            simpleRender(startLayer, endLayer);
            break;
        case RenderBuffer::Sprite:
            spriteRender(startLayer, endLayer);
            break;
        case RenderBuffer::Text:
            textRender(startLayer, endLayer);
            break;
        default:
            break;
        }
    };

    for(uint8_t i = 0; i < 10; i++)
    {
        bool noSimple = m_ObjectBuffer.getLayerSize(i) == 0;
        bool noSprite = m_SpriteBuffer.getLayerSize(i) == 0;
        bool noText   = m_TextObjBuffer.getLayerSize(i) == 0;
        if(i != 9)
        {
            if(noSprite && noText)
            {
                if(currentBuffer != RenderBuffer::Simple)
                {
                    renderCurrentBuffer(currentBuffer, startLayer, i - 1);
                    startLayer    = i;
                    currentBuffer = RenderBuffer::Simple;
                }
            }
            else if(noSimple && noText)
            {
                if(currentBuffer != RenderBuffer::Sprite)
                {
                    renderCurrentBuffer(currentBuffer, startLayer, i - 1);
                    startLayer    = i;
                    currentBuffer = RenderBuffer::Sprite;
                }
            }
            else if(noSimple && noSprite)
            {
                if(currentBuffer != RenderBuffer::Text)
                {
                    renderCurrentBuffer(currentBuffer, startLayer, i - 1);
                    startLayer    = i;
                    currentBuffer = RenderBuffer::Text;
                }
            }
            else
            {
                renderCurrentBuffer(currentBuffer, startLayer, i - 1);
                if(!noSimple)
                    simpleRender(i, i);
                if(!noSprite)
                    spriteRender(i, i);
                if(!noText)
                    textRender(i, i);
                currentBuffer = RenderBuffer::None;
            }
        }
        else
        {
            renderCurrentBuffer(currentBuffer, startLayer, i - 1);
            if(!noSimple)
                simpleRender(i, i);
            if(!noSprite)
                spriteRender(i, i);
            if(!noText)
                textRender(i, i);
            currentBuffer = RenderBuffer::None;
        }
    }

    // Resets settings
    orderBuffersByYAxisSetting = false;

    m_ObjectBuffer.clear();
    m_SpriteBuffer.clear();
    m_TextObjBuffer.clear();
}

void Render::simpleRender(uint8_t startLayer, uint8_t endLayer)
{
    if(!m_VertexBuffer->isEmpty())   // If the buffer is not empty, it empties it
    {
        m_VertexBuffer->clearBufferData();
        Log::warning("Vertex Buffer was not empty!");
    }

    m_SimpleShader->bind();
    // Goes through all the objects in the buffer and renders them
    for(uint32_t i = m_ObjectBuffer.getLayerPos(startLayer); i < m_ObjectBuffer.getLayerPos(endLayer + 1); i++)
    {
        ColouredObject *obj = m_ObjectBuffer[i];

        auto vertices = obj->convertToColouredVertices();
```

```cpp
        // Checks if the buffer is full or the buffer is too big and draws what there is
        if(!m_VertexBuffer->canStore(obj->getSizeOfVertices()))
        {
            draw(*m_SimpleVAO);
            m_VertexBuffer->clearBufferData();   // Resets the buffer so it can draw again
        }
        // Adds the current object to the buffer by creating its quad (this is for memory efficiency)
        m_VertexBuffer->addToBuffer((void *) &vertices, obj->getSizeOfVertices());
        delete obj;
    }

    if(!m_VertexBuffer->isEmpty())   // If the buffer is not empty, it empties it
    {
        draw(*m_SimpleVAO);
        m_VertexBuffer->clearBufferData();
    }
}

void Render::spriteRender(uint8_t startLayer, uint8_t endLayer)
{
    if(!m_VertexBuffer->isEmpty())   // If the buffer is not empty, it empties it
    {
        m_VertexBuffer->clearBufferData();
        Log::warning("Vertex Buffer was not empty!");
    }

    m_SpriteShader->bind();
    uint8_t currentTexSlot = 0;   // This stores the slot the current texture is bound to, so it can set the texID
     part of the vertex
    Texture::clearBufferSlots();

    for(uint32_t i = m_SpriteBuffer.getLayerPos(startLayer); i < m_SpriteBuffer.getLayerPos(endLayer + 1); i++)
    {
        TexturedObject *obj = m_SpriteBuffer[i];
        // Checks if the buffer is full or the buffer is too big and draws what there is
        if(!m_VertexBuffer->canStore(obj->getSizeOfVertices()))
        {
            draw(*m_SpriteVAO);
            m_VertexBuffer->clearBufferData();   // Resets the buffer so it can draw again
            Texture::clearBufferSlots();
            currentTexSlot = 0;   // resets this as all the textures have been rendered
        }

        // Gets the texture slot
        uint8_t texSlot = Texture::getBoundSlot(Sprite::getSprite(obj->spriteID)->getTexture());
        if(texSlot == 32)   // This means the texture is not bound, so it gets bound
        {
            if(currentTexSlot == 32)
            {
                draw(*m_SpriteVAO);
                m_VertexBuffer->clearBufferData();   // Resets the buffer so it can draw again
                Texture::clearBufferSlots();
                currentTexSlot = 0;   // resets this as all the textures have been rendered
            }
            texSlot = currentTexSlot;
            currentTexSlot++;
            Sprite::getSprite(obj->spriteID)->bind(texSlot);
        }

        if(texSlot >= currentTexSlot)
        {
            Log::warning("Unbounded texSlot given!");
        }

        auto vertices = obj->convertToTexturedVertices(texSlot);   // Creates the vertices

        // Adds the current object to the buffer by creating its quad (this is for memory efficiency)
        m_VertexBuffer->addToBuffer((void *) &vertices, obj->getSizeOfVertices());

        delete obj;   // Deletes the object
    }
    if(!m_VertexBuffer->isEmpty())   // If the buffer is not empty, it empties it
    {
        draw(*m_SpriteVAO);
        m_VertexBuffer->clearBufferData();
    }
}

void Render::textRender(uint8_t startLayer, uint8_t endLayer)
{
    if(!m_VertexBuffer->isEmpty())   // If the buffer is not empty, it empties it
    {
        m_VertexBuffer->clearBufferData();
        Log::warning("Vertex Buffer was not empty!");
    }

    m_TextShader->bind();
    uint8_t currentTexSlot = 0;   // This stores the slot the current texture is bound to
    Texture::clearBufferSlots();
    for(uint32_t i = m_TextObjBuffer.getLayerPos(startLayer); i < m_TextObjBuffer.getLayerPos(endLayer + 1); i++)
    {
        TextObject *text = m_TextObjBuffer[i];

        float xOffset = 0.0f;
        if(text->text.empty())
            continue;

        // This goes through each character in the text and adds it to the buffer
        for(std::string::const_iterator c = text->text.begin(); c != text->text.end(); c++)
        {
            Character *ch = &characters[*c];

            // Checks if the buffer can store the new vertices
            if(!m_VertexBuffer->canStore(text->getSizeOfVertices()))
            {
                draw(*m_SpriteVAO);
                m_VertexBuffer->clearBufferData();   // Resets the buffer so it can draw again
                Texture::clearBufferSlots();
                currentTexSlot = 0;   // resets this as all the textures have been rendered
            }

            // Gets the textures slot
            uint8_t texSlot = Texture::getBoundSlot(ch->texture);
            if(texSlot == 32)   // This means it is not bound so it gets bound
            {
                if(currentTexSlot == 32)   // This means the it cannot render anymore textures
                {
                    draw(*m_TextVAO);
                    m_VertexBuffer->clearBufferData();   // Resets the buffer so it can draw again
                    Texture::clearBufferSlots();
                    currentTexSlot = 0;   // resets this as all the textures have been rendered
                }
                texSlot = currentTexSlot;
                currentTexSlot++;
                ch->texture->bind(texSlot);
            }

            // Gets the vertices
            auto vertices = text->convertCharacterToVertices(ch, xOffset, texSlot);

            m_VertexBuffer->addToBuffer((const void *) &vertices, text->getSizeOfVertices());

            // now advance cursors for next glyph (note that advance is number of 1/64 pixels)
            float newScale = text->scale / 100;
            xOffset += (ch->advance >> 6) * newScale;   // bitshift by 6 to get value in pixels (2^6 = 64)
        }

        // Deletes the text obj
        delete text;
    }
    if(!m_VertexBuffer->isEmpty())   // If the buffer is not empty, it empties it
    {
        draw(*m_TextVAO);
        m_VertexBuffer->clearBufferData();
    }
}

void Render::draw(VertexArray &vao) const   // Assumes VAO and shader have already been bound
{
    // Binds what this renderer is using for vertices
    vao.bind();
    m_VertexBuffer->bind();
    m_IndexBuffer->bind();
    GLCall(glDrawElements(GL_TRIANGLES, m_IndexBuffer->getCount(), GL_UNSIGNED_INT, nullptr));
    vao.unbind();
}

void Render::spriteImpl(float x, float y, double rotation, float width, float height, Sprite::ID spriteID, uint8_t
     layer, bool isOverlay)
{
    // This creates a collision box for the sprite, so it can check if it is in frame
    CollisionBox box = {{-width / 2, -height / 2}, {width / 2, height / 2}};
    if(isOverlay || Application::isInFrame(x, y, box))
    {
        // Creates an object to store the information and adds it to the buffer through the function which takes settings into account
        TexturedObject *obj = new TexturedObject({x, y}, width, height, rotation, true, spriteID);
        m_SpriteBuffer.addElement(obj, layer, orderBuffersByYAxisSetting);
    }
}

void Render::textImpl(std::string &text, float x, float y, float scale, glm::vec4 colour, uint8_t layer, bool isCentered, bool isOverlay)   // NOTE: Scale is a percentage
{
    // Gets the collision box for the text to check if it is frame
    CollisionBox box = getTextCollisionBox(text, scale);

    if(isOverlay || Application::isInFrame(x, y, box))
    {
        // Creates an object to store the information and adds it to the buffer through the function which takes settings into account
        TextObject *obj = new TextObject(text, scale, {x, y}, box.upperBound.x - box.lowerBound.x, box.upperBound.y - box.lowerBound.y, 0.0f, colour, isCentered);
        m_TextObjBuffer.addElement(obj, layer, orderBuffersByYAxisSetting);
    }
}

void Render::hoverTextImpl(std::string &inpText, float x, float y, float scale, glm::vec4 textColour, glm::vec4 backgroundColour, uint8_t layer, bool isOverlay)
{
    CollisionBox box = Render::getTextCollisionBox(inpText, scale);

    float borderWidth = 2.0f;

    float width  = box.upperBound.x - box.lowerBound.x;
    float height = box.upperBound.y - box.lowerBound.y;

    float yOffset = 4.0f;

    float textX = x + box.lowerBound.x;
    float textY = y + yOffset + borderWidth - box.lowerBound.y + height / 2;

    rectangle(x, y + yOffset + borderWidth + height / 2, 0.0f, width + 2 * borderWidth, height + 2 * borderWidth, backgroundColour, layer, true, isOverlay);
    text(inpText, x, textY, scale, textColour, layer, true, isOverlay);
}

void Render::rectangleImpl(float x, float y, double rotation, float width, float height, glm::vec4 colour, uint8_t layer, bool isCentered, bool isOverlay)
{
    // Creates a collision box, taking into account if it wants to be centered or not
    CollisionBox box;
    if(isCentered)
        box = {{-width / 2, -height / 2}, {width / 2, height / 2}};
    else
        box = {{0, 0}, {width, height}};

    if(isOverlay || Application::isInFrame(x, y, box))
    {
        // Creates an object to store the information
        ColouredObject *obj = new ColouredObject({x, y}, width, height, rotation, isCentered, colour);
        m_ObjectBuffer.addElement(obj, layer, orderBuffersByYAxisSetting);
    }
}

// This is for rendering a box with a border
void Render::rectangleImpl(float x, float y, float width, float height, glm::vec4 colour, float borderWidth, glm::vec4 borderColour, uint8_t layer, bool isCentered, bool isOverlay)
{
    float tempX, tempY;   // Stores the x and y position of the botton corner of the rectangle
    if(isCentered)
    {
        tempX = x - width / 2;
        tempY = y - height / 2;
    }
    else
    {
        tempX = x;
        tempY = y;
    }
    // Adds the normal rectangle to the buffer
    rectangle(x, y, 0.0f, width, height, colour, layer, isCentered, isOverlay);

    // For each border it renders more rectangles
    rectangle(tempX, tempY, 0.0f, width, borderWidth, borderColour, layer, false, isOverlay);
    rectangle(tempX, tempY, 0.0f, borderWidth, height, borderColour, layer, false, isOverlay);
    rectangle(tempX, tempY + height - borderWidth, 0.0f, width, borderWidth, borderColour, layer, false, isOverlay
    );
    rectangle(tempX + width - borderWidth, tempY, 0.0f, borderWidth, height, borderColour, layer, false, isOverlay
    );
}

float Render::getTextWidthImpl(std::string &text, float scale)
{
    // This goes through each letter of text and adds all the advance together to get the width
    float textWidth = 0;
    float newScale  = scale / 100;
    for(std::string::const_iterator c = text.begin(); c != text.end(); c++)
    {
        Character *ch = &characters[*c];
        textWidth += (ch->advance >> 6) * newScale;
    }
    return textWidth;
}

float Render::getTextHeightImpl(std::string &text, float scale)
{
    // This goes through each letter of the text and finds the character with the biggest height and returns that
    float textHeight = 0;
    float minYPos    = 0;
    float newScale   = scale / 100;

    for(std::string::const_iterator c = text.begin(); c != text.end(); c++)
    {
        Character *ch = &characters[*c];
        float      h  = ch->bearing.y * newScale;
        if(h > textHeight)
            textHeight = h;

        float y = (ch->bearing.y - ch->size.y) * newScale;
        if(y < minYPos)
            minYPos = y;
    }

    return textHeight - minYPos;
}

CollisionBox Render::getTextCollisionBoxImpl(std::string &text, float scale)
{
    // This does the same as the above two function but creates a collision box and does it with one for loop
    float textWidth  = 0;
    float textHeight = 0;
    float minYPos    = 0;
    float newScale   = scale / 100;

    for(std::string::const_iterator c = text.begin(); c != text.end(); c++)
    {
        Character *ch = &characters[*c];
        textWidth += (ch->advance >> 6) * newScale;
        float h = ch->bearing.y * newScale;
        if(h > textHeight)
            textHeight = h;
        float y = (ch->bearing.y - ch->size.y) * newScale;
        if(y < minYPos)
            minYPos = y;
    }

    return {{0.0f, minYPos}, {textWidth, textHeight}};
}

void Render::orderBuffersByYAxisImpl()
{
    // This sets the setting to true, and does a quick check to see if all the buffers are empty
    if(m_SpriteBuffer.size() != 0 || m_TextObjBuffer.size() != 0 || m_ObjectBuffer.size() != 0)
        Log::critical("Turning on setting 'orderBuffersByYAxis' after buffers have started to be filled!", LOGINFO
    );
    orderBuffersByYAxisSetting = true;
}
```

3.4.1 Effects

rendering/effect/Effect.h

```
#pragma once


namespace Effect
{
    // This stores all the effect types there could be

    class Effect
    {
      public:
        enum Type
        {
            shaderEffect,
            removeShaderEffect,
            objectParticleSpawner
        };

      public:
        Effect()
        {
        }
        virtual ~Effect() {}

        virtual Type getType() const = 0;
    };

}   // namespace Effect
```

../src/rendering/effect/Effect.h

rendering/effect/ShaderEffectCarrier.h

```
#include "Effect.h"

namespace Effect
{
    // These are for all the different shader effects that could be applied
    class ShaderEffectCarrier : public Effect
    {
      protected:
        uint16_t m_ID;

      public:
        ShaderEffectCarrier(uint16_t id)
            : m_ID(id)
        {
        }
        virtual ~ShaderEffectCarrier() override {}

        uint16_t             getID() { return m_ID; }
        virtual Effect::Type getType() const override { return Effect::Type::shaderEffect; }
    };

    // This is for removing an effect from a layer
    class RemoveShaderEffect : public ShaderEffectCarrier
    {
      public:
        RemoveShaderEffect(uint16_t id)
            : ShaderEffectCarrier(id)
        {
        }
        virtual ~RemoveShaderEffect() override {}

        virtual Effect::Type getType() const override { return Effect::Type::removeShaderEffect; }
    };
}   // namespace Effect
```

../src/rendering/effect/ShaderEffectCarrier.h

rendering/effect/ShaderEffectsManager.h

```
#pragma once

#include <string>

#include "Log.h"
#include "Shader.h"
#include "ShaderEffectCarrier.h"
```

```
namespace Effect
{
    class ShaderEffect
    {
      public:
        enum class Type
        {
            normal,
            includeOverlay,
            onlyOverlay
        };

      protected:
        const std::string m_Name;

        bool simpleShader, spriteShader, textShader;

        Type m_Type;

      public:
        ShaderEffect(const std::string &name, Type type, bool simpleShader, bool spriteShader, bool textShader)
            : m_Name(name), m_Type(type), simpleShader(simpleShader), spriteShader(spriteShader), textShader(
    textShader)
        {
        }
        virtual ~ShaderEffect() {}

        std::string  getName() { return m_Name; }
        Type         getType() { return m_Type; }
        virtual void setEffect(Shader &s) const = 0;

        bool forSimpleShader() { return simpleShader; }
        bool forSpriteShader() { return spriteShader; }
        bool forTextShader() { return textShader; }
    };

    class UniformVec4 : public ShaderEffect
    {
      protected:
        glm::vec4 vec;

      public:
        UniformVec4(const std::string &name, glm::vec4 vec, Type type, bool simpleShader, bool spriteShader, bool
    textShader)
            : ShaderEffect(name, type, simpleShader, spriteShader, textShader), vec(vec)
        {
        }
        virtual ~UniformVec4() override {}

        virtual void setEffect(Shader &s) const override
        {
            s.setUniform4f(m_Name, vec[0], vec[1], vec[2], vec[3]);
        }

        void setVec(glm::vec4 newVec) { vec = newVec; }
    };

    class UniformMat4 : public ShaderEffect
    {
      protected:
        glm::mat4 mat;

      public:
        UniformMat4(const std::string &name, glm::mat4 mat, Type type, bool simpleShader, bool spriteShader, bool
    textShader)
            : ShaderEffect(name, type, simpleShader, spriteShader, textShader), mat(mat)
        {
        }
        virtual ~UniformMat4() override {}

        virtual void setEffect(Shader &s) const override
        {
            s.setUniformMat4f(m_Name, mat);
        }
        void setMat(glm::mat4 newMat) { mat = newMat; }
    };

    // This is the class for managing the shader effects, their IDs and storing and sending the effects that are
    carried
    class ShaderEffectsManager
    {
      public:
        ShaderEffectsManager(const ShaderEffectsManager &) = delete;
        ~ShaderEffectsManager();

        static uint16_t sendShaderEffect(const std::string &s, glm::vec4 vec, bool simpleShader = true, bool
    spriteShader = true, bool textShader = true, bool includeOverlay = false)
        {
            return get().sendShaderEffectImpl(s, vec, simpleShader, spriteShader, textShader, includeOverlay);
        }
        static uint16_t sendShaderEffect(const std::string &s, glm::mat4 mat, bool simpleShader = true, bool
    spriteShader = true, bool textShader = true, bool includeOverlay = false)
        {
            return get().sendShaderEffectImpl(s, mat, simpleShader, spriteShader, textShader, includeOverlay);
        }
        static uint16_t sendOverlayEffect(const std::string &s, glm::vec4 vec, bool simpleShader = true, bool
    spriteShader = true, bool textShader = true)
        {
            return get().sendOverlayEffectImpl(s, vec, simpleShader, spriteShader, textShader);
        }
        static uint16_t sendOverlayEffect(const std::string &s, glm::mat4 mat, bool simpleShader = true, bool
    spriteShader = true, bool textShader = true)
        {
            return get().sendOverlayEffectImpl(s, mat, simpleShader, spriteShader, textShader);
        }

        static void          deleteShaderEffect(uint16_t id) { get().deleteShaderEffectImpl(id); }
        static ShaderEffect *getShaderEffect(uint16_t id) { return get().getShaderEffectImpl(id); }

        static uint16_t findShaderEffect(const std::string &s) { return get().findShaderEffectImpl(s); }

        static void updateShaderEffects() { get().updateShaderEffectsImpl(); }

        static ShaderEffectsManager &get()
        {
            static ShaderEffectsManager s_Instance;
            return s_Instance;
        }

      private:
        static ShaderEffectsManager s_Instance;
        std::vector<ShaderEffect *> m_Effects;

        ShaderEffectsManager();

        uint16_t      sendShaderEffectImpl(const std::string &s, glm::vec4 vec, bool simpleShader, bool
    spriteShader, bool textShader, bool includeOverlay);
        uint16_t      sendShaderEffectImpl(const std::string &s, glm::mat4 mat, bool simpleShader, bool
    spriteShader, bool textShader, bool includeOverlay);
        uint16_t      sendOverlayEffectImpl(const std::string &s, glm::vec4 vec, bool simpleShader, bool
    spriteShader, bool textShader);
        uint16_t      sendOverlayEffectImpl(const std::string &s, glm::mat4 mat, bool simpleShader, bool
    spriteShader, bool textShader);
        void          deleteShaderEffectImpl(uint16_t id);
        ShaderEffect *getShaderEffectImpl(uint16_t id);
        uint16_t      findShaderEffectImpl(const std::string &s);

        void updateShaderEffectsImpl();
    };
}   // namespace Effect
```

../src/rendering/effect/ShaderEffectsManager.h

rendering/effect/ShaderEffectsManager.cpp

```
#include "ShaderEffectsManager.h"

#include "Application.h"
#include <GLM.h>

#include <vector>

namespace Effect
{
    ShaderEffectsManager::ShaderEffectsManager()
    {
    }

    ShaderEffectsManager::~ShaderEffectsManager()
    {
```

```
    for(ShaderEffect *s : m_Effects)
        delete s;
}

// These functions handle the sending of an effect, by creating them, adding them to the cache and sending them through the layers
uint16_t ShaderEffectsManager::sendShaderEffectImpl(const std::string &s, glm::vec4 vec, bool simpleShader, bool spriteShader, bool textShader, bool includeOverlay)
{
    ShaderEffect::Type type;
    if(includeOverlay)
        type = ShaderEffect::Type::includeOverlay;
    else
        type = ShaderEffect::Type::normal;

    UniformVec4 *e = new UniformVec4(s, vec, type, simpleShader, spriteShader, textShader);
    m_Effects.push_back(e);

    ShaderEffectCarrier messenger((uint16_t) m_Effects.size());
    Application::setEffect(&messenger, includeOverlay);

    return messenger.getID();
}

uint16_t ShaderEffectsManager::sendShaderEffectImpl(const std::string &s, glm::mat4 mat, bool simpleShader, bool spriteShader, bool textShader, bool includeOverlay)
{
    ShaderEffect::Type type;
    if(includeOverlay)
        type = ShaderEffect::Type::includeOverlay;
    else
        type = ShaderEffect::Type::normal;

    UniformMat4 *e = new UniformMat4(s, mat, type, simpleShader, spriteShader, textShader);
    m_Effects.push_back(e);

    ShaderEffectCarrier messenger((uint16_t) m_Effects.size());
    Application::setEffect(&messenger, includeOverlay);

    return messenger.getID();
}

// These send the effects through the overlays only and not all the layers
uint16_t ShaderEffectsManager::sendOverlayEffectImpl(const std::string &s, glm::vec4 vec, bool simpleShader, bool spriteShader, bool textShader)
{
    ShaderEffect::Type type = ShaderEffect::Type::onlyOverlay;

    UniformVec4 *e = new UniformVec4(s, vec, type, simpleShader, spriteShader, textShader);
    m_Effects.push_back(e);

    ShaderEffectCarrier messenger((uint16_t) m_Effects.size());
    Application::setOverlayEffect(&messenger);

    return messenger.getID();
}

uint16_t ShaderEffectsManager::sendOverlayEffectImpl(const std::string &s, glm::mat4 mat, bool simpleShader, bool spriteShader, bool textShader)
{
    ShaderEffect::Type type = ShaderEffect::Type::onlyOverlay;

    UniformMat4 *e = new UniformMat4(s, mat, type, simpleShader, spriteShader, textShader);
    m_Effects.push_back(e);

    ShaderEffectCarrier messenger((uint16_t) m_Effects.size());
    Application::setOverlayEffect(&messenger);

    return messenger.getID();
}

// This manages deleting a shader effect from all the layers and its storage
void ShaderEffectsManager::deleteShaderEffectImpl(uint16_t id)
{
    if(id > m_Effects.size() || id < 1)
    {
        Log::warning("Tried to delete effect outside of range");
        return;
    }
```

```
        RemoveShaderEffect messenger(id);
        Application::setEffect(&messenger, true);

        delete m_Effects[id - 1];
        m_Effects.erase(m_Effects.begin() + id - 1);
    }

    ShaderEffect *ShaderEffectsManager::getShaderEffectImpl(uint16_t id)
    {
        if(id > m_Effects.size() || id < 1)
        {
            Log::warning("Tried to access effect outside of range");
            return nullptr;
        }
        return m_Effects[id - 1];
    }

    // This finds the id of a shader by use of its name
    uint16_t ShaderEffectsManager::findShaderEffectImpl(const std::string &s)
    {
        for(uint16_t i = 0; i < m_Effects.size(); i++)
        {
            if(s == m_Effects[i]->getName())
                return i + 1;
        }
        Log::warning("Did not find shader Effect!");
        return 0;
    }

    void ShaderEffectsManager::updateShaderEffectsImpl()
    {
        for(uint16_t i = 0; i < m_Effects.size(); i++)
        {
            ShaderEffectCarrier messenger(i + 1);
            switch(m_Effects[i]->getType())
            {
            case ShaderEffect::Type::normal:
                Application::setEffect(&messenger, false);
                break;
            case ShaderEffect::Type::includeOverlay:
                Application::setEffect(&messenger, true);
                break;
            case ShaderEffect::Type::onlyOverlay:
                Application::setOverlayEffect(&messenger);
                break;
            default:
                Log::warning("Unknown effect type");
            }
        }
    }
}   // namespace Effect
```

../src/rendering/effect/ShaderEffectsManager.cpp

3.4.2 OpenGL Interface

rendering/glInterface/glDebug.h

```
#pragma once

#include <GL/glew.h>

#include "Log.h"
#include <sstream>

// This creates an ASSERT definition so that when debugging I can set a breakpoint if a statement does not conform
#ifdef IS_ON_WINDOWS
    #define ASSERT(x) \
        if(!(x)) __debugbreak();
#else
    #define ASSERT(x) \
        if(!(x)) __builtin_trap();
#endif

// This uses an old method of debugging openGL by making checking for errors after every line
#define GLCall(x)   \
    GLClearError(); \
    x;              \
    ASSERT(GLLogCall(#x, __FILE__, __LINE__))
```

```
static void GLClearError()
{
    // This goes through every current error and removes it
    while(glGetError() != GL_NO_ERROR)
        ;
}

static bool GLLogCall(const char *function, const char *file, int line)
{
    // This goes through all the errors Logs them
    while(GLenum error = glGetError())
    {
        std::stringstream ss;
        ss << "OpenGL error (" << error << "): " << function;
        Log::critical(ss.str().c_str(), file, line);
        return false;
    }
    return true;
}
```

../src/rendering/glInterface/glDebug.h

rendering/glInterface/IndexBuffer.h

```
#pragma once

// This class is for storing and creating the index buffer so that that I do not have to have duplicate vertices
    in the vertex buffer
class IndexBuffer
{
  private:
    uint32_t m_RendererID;
    uint32_t m_Count;

  public:
    IndexBuffer(const uint32_t *data, uint32_t count);
    IndexBuffer(uint32_t count);
    ~IndexBuffer();

    void bind() const;
    void unbind() const;

    inline uint32_t getCount() const { return m_Count; }
};
```

../src/rendering/glInterface/IndexBuffer.h

rendering/glInterface/IndexBuffer.cpp

```
#include "glDebug.h"

#include "IndexBuffer.h"

#include "Renderer.h"

IndexBuffer::IndexBuffer(const uint32_t *data, uint32_t count)
    : m_Count(count)
{
    // This will generate the buffer and add the data
    GLCall(glGenBuffers(1, &m_RendererID));
    GLCall(glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_RendererID));
    GLCall(glBufferData(GL_ELEMENT_ARRAY_BUFFER, count * sizeof(uint32_t), data, GL_STATIC_DRAW));
}

IndexBuffer::IndexBuffer(uint32_t count)
    : m_Count(count)
{
    // This generates a default for the data with a given count of vertices
    uint32_t *data = new uint32_t[count];

    uint32_t squares = count / 6;
    for(uint32_t i = 0; i < squares; i++)
    {
        data[i * 6]     = (i * 4);
        data[i * 6 + 1] = (i * 4) + 1;
        data[i * 6 + 2] = (i * 4) + 2;
        data[i * 6 + 3] = (i * 4) + 2;
        data[i * 6 + 4] = (i * 4) + 3;
        data[i * 6 + 5] = (i * 4);
    }
    GLCall(glGenBuffers(1, &m_RendererID));
    GLCall(glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_RendererID));
```

```cpp
    GLCall(glBufferData(GL_ELEMENT_ARRAY_BUFFER, m_Count * sizeof(uint32_t), data, GL_STATIC_DRAW));

    delete[] data;   // Makes sure to delete the data
}

IndexBuffer::~IndexBuffer()
{
    // Deletes the buffer
    GLCall(glDeleteBuffers(1, &m_RendererID));
}

void IndexBuffer::bind() const
{
    // Binds the buffer
    GLCall(glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_RendererID));
}

void IndexBuffer::unbind() const
{
    // Unbinds the buffer
    GLCall(glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0));
}
```

../src/rendering/glInterface/IndexBuffer.cpp

rendering/glInterface/Shader.h

```cpp
#pragma once

#include <GLM.h>
#include <string>
#include <tuple>
#include <unordered_map>

// This class is for interacting with the shader
class Shader
{
  private:
    std::string                          m_FilePath;   // This is for debugging purposes
    uint32_t                             m_RendererID;
    std::unordered_map<std::string, int> m_UniformLocationCache;   // This stores the uniform locations so they
    can be easily changed

    // Internal functions for the shader
    int                                  getUniformLocation(const std::string &name);
    std::tuple<std::string, std::string> parseShader(const std::string &filepath);
    uint32_t                             compileShader(uint32_t type, const std::string &source);
    int                                  createShader(const std::string &vertexShader, const std::string &
    fragmentShader);

  public:
    Shader(const std::string &filepath);
    ~Shader();

    void bind() const;
    void unbind() const;

    // Functions for setting the variables in the shader;
    void setUniform1i(const std::string &name, int value);
    void setUniform1iv(const std::string &name, int count, const int *value);
    void setUniform1f(const std::string &name, float value);
    void setUniform4f(const std::string &name, float v0, float v1, float f2, float f3);
    void setUniformMat4f(const std::string &name, const glm::mat4 &matrix);
    void setUniformMat2f(const std::string &name, const glm::mat2 &matrix);
};
```

../src/rendering/glInterface/Shader.h

rendering/glInterface/Shader.cpp

```cpp
#include "glDebug.h"

#include "Shader.h"

#include <fstream>
#include <iostream>
#include <sstream>
#include <string>

#include "Log.h"
#include "Renderer.h"
```

```cpp
Shader::Shader(const std::string &filepath)
    : m_FilePath(filepath), m_RendererID(0)
{
    Log::variable("Loading shader", filepath);
    // Reads the shader file and splits them into their two types
    auto [vertexShader, fragmentShader] = parseShader(filepath);
    // Compiles the shader files
    m_RendererID                        = createShader(vertexShader, fragmentShader);
}
Shader::~Shader()
{
    GLCall(glDeleteProgram(m_RendererID));
}

int Shader::createShader(const std::string &vertexShader, const std::string &fragmentShader)
{
    // Creates the program the shaders will link to
    GLCall(uint32_t program = glCreateProgram());
    // Compiles the each shader
    uint32_t vs = compileShader(GL_VERTEX_SHADER, vertexShader);
    uint32_t fs = compileShader(GL_FRAGMENT_SHADER, fragmentShader);

    GLCall(glAttachShader(program, vs));
    GLCall(glAttachShader(program, fs));
    GLCall(glLinkProgram(program));
    GLCall(glValidateProgram(program));

    // Checks if the program linked correctly
    GLint isLinked = 0;
    glGetProgramiv(program, GL_LINK_STATUS, (int *) &isLinked);
    if(isLinked == GL_FALSE)
    {
        // If there was an error, it gets the information and logs it
        GLint maxLength = 0;
        glGetProgramiv(program, GL_INFO_LOG_LENGTH, &maxLength);

        // The maxLength includes the NULL character
        std::vector<GLchar> infoLog(maxLength);
        glGetProgramInfoLog(program, maxLength, &maxLength, &infoLog[0]);

        // We don't need the program anymore.
        glDeleteProgram(program);
        // Don't leak shaders either.
        glDeleteShader(vs);
        glDeleteShader(fs);

        Log::error(infoLog.data(), LOGINFO);
        Log::critical("Shader link failure!", LOGINFO);
    }

    glDetachShader(program, vs);
    glDetachShader(program, fs);

    GLCall(glDeleteShader(vs));
    GLCall(glDeleteShader(fs));

    return program;
}

uint32_t Shader::compileShader(uint32_t type, const std::string &source)
{
    // Compiles and generates an id for the shader
    uint32_t     id  = glCreateShader(type);
    const char * src = source.c_str();
    glShaderSource(id, 1, &src, nullptr);
    glCompileShader(id);

    // Checks for any errors when compiling
    int result;
    glGetShaderiv(id, GL_COMPILE_STATUS, &result);
    if(result == GL_FALSE)
    {
        // Gets the information and logs it
        int length;
        glGetShaderiv(id, GL_INFO_LOG_LENGTH, &length);
        char *message = (char *) alloca(length * sizeof(char));
        glGetShaderInfoLog(id, length, &length, message);
        std::stringstream ss;
        ss << "Failed to compile "
           << (type == GL_VERTEX_SHADER ? "vertex" : "fragment")
```

```
            << " shader! "
            << message << std::endl;
        Log::critical(ss.str().c_str(), LOGINFO);
        glDeleteShader(id);
        return 0;
    }

    return id;
}

std::tuple<std::string, std::string> Shader::parseShader(const std::string &filepath)
{
    // Checks if the file exists
    std::ifstream stream(filepath);
    if(!stream.good())
    {
        Log::error("Shader file not found!", LOGINFO);
        Log::variable("Shader filename", filepath);
        return {"", ""};
    }

    enum class ShaderType
    {
        NONE     = -1,
        VERTEX   = 0,
        FRAGMENT = 1
    };

    // Reads each line from the file
    std::string       line;
    std::stringstream ss[2];
    ShaderType        type = ShaderType::NONE;
    while(getline(stream, line))
    {
        if(line.find("#shader") != std::string::npos)   // Checks to find the label for which shader it it
        {
            // Sets the type to the current shader labelled
            if(line.find("vertex") != std::string::npos)
                type = ShaderType::VERTEX;
            else if(line.find("fragment") != std::string::npos)
                type = ShaderType::FRAGMENT;
        }
        else
        {
            // Checks to see if the shader type is set up
            if(type == ShaderType::NONE)
            {
                // Logs a warning and ignores the line
                Log::warning("Shader file not set up correctly");
                continue;
            }
            // Adds the line to the stream
            ss[(int) type] << line << '\n';
        }
    }

    // Returns both the vertex and fragment shaders
    return {ss[(int) ShaderType::VERTEX].str(), ss[(int) ShaderType::FRAGMENT].str()};
}

void Shader::bind() const
{
    GLCall(glUseProgram(m_RendererID));
}

void Shader::unbind() const
{
    GLCall(glUseProgram(0));
}

// For each type of variable it will find the location of the uniform and call the relevent function
void Shader::setUniform1i(const std::string &name, int value)
{
    bind();
    GLCall(glUniform1i(getUniformLocation(name), value));
}

void Shader::setUniform1iv(const std::string &name, int count, const int *value)
{
    bind();
    GLCall(glUniform1iv(getUniformLocation(name), count, value));
}

void Shader::setUniform1f(const std::string &name, float value)
{
    bind();
    GLCall(glUniform1f(getUniformLocation(name), value));
}

void Shader::setUniform4f(const std::string &name, float v0, float v1, float v2, float v3)
{
    bind();
    GLCall(glUniform4f(getUniformLocation(name), v0, v1, v2, v3));
}
void Shader::setUniformMat2f(const std::string &name, const glm::mat2 &matrix)
{
    bind();
    GLCall(glUniformMatrix2fv(getUniformLocation(name), 1, GL_FALSE, &matrix[0][0]));
}

void Shader::setUniformMat4f(const std::string &name, const glm::mat4 &matrix)
{
    bind();
    GLCall(glUniformMatrix4fv(getUniformLocation(name), 1, GL_FALSE, &matrix[0][0]));
}

int Shader::getUniformLocation(const std::string &name)
{
    // If the uniform has already been found, it will return that
    if(m_UniformLocationCache.find(name) != m_UniformLocationCache.end())
        return m_UniformLocationCache[name];

    // Otherwise it will get the location of the uniform
    GLCall(int location = glGetUniformLocation(m_RendererID, name.c_str()));
    if(location == -1)   // -1 is returned when that uniform does not exist
    {
        std::stringstream ss;
        ss << "Uniform '"
           << name << "' doesn't exist!";
        Log::warning(ss.str().c_str());
    }
    // Adds the name to the map so it can be easily gained next time
    m_UniformLocationCache[name] = location;
    return location;
}
```

../src/rendering/glInterface/Shader.cpp

rendering/glInterface/Texture.h

```
#pragma once

#include <string>

// This is for storing and loading textures
class Texture
{
  private:
    // This is a cache for what is currently bound to which slot
    static const Texture *bufferStorage[32];

    uint32_t       m_RendererID;
    std::string    m_FilePath;   // This is for debugging purposes
    unsigned char *m_LocalBuffer;
    int            m_Width, m_Height, m_BPP;

  public:
    Texture(const std::string &path);
    Texture(uint16_t width, uint16_t height, unsigned char *buffer);
    ~Texture();

    void bind(uint8_t slot = 0) const;
    void unbind() const;

    inline int getWidth() const { return m_Width; }
    inline int getHeight() const { return m_Height; }
    inline uint32_t getID() const { return m_RendererID; }

    // Functions for interacting with the bufferStorage cache
    static const Texture *getTextureInBuffer(uint8_t slot) { return bufferStorage[slot]; }
    static uint8_t        getBoundSlot(Texture *tex);
    static void             clearBufferSlots();
};
```

../src/rendering/glInterface/Texture.h

rendering/glInterface/Texture.cpp

```
#include "glDebug.h"

#include "Texture.h"

#include "Log.h"
#include "Renderer.h"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image/stb_image.h"

const Texture *Texture::bufferStorage[32];

Texture::Texture(const std::string &path)
    : m_RendererID(0), m_FilePath(path), m_LocalBuffer(nullptr), m_Width(0), m_Height(0), m_BPP(0)
{
    // This is for loading a texture from a file using stb_image
    stbi_set_flip_vertically_on_load(1);    // This will it when loading so positive y goes up the image
    m_LocalBuffer = stbi_load(path.c_str(), &m_Width, &m_Height, &m_BPP, 4);

    GLCall(glGenTextures(1, &m_RendererID));
    GLCall(glBindTexture(GL_TEXTURE_2D, m_RendererID));

    // Loads the image into the buffer
    GLCall(glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, m_Width, m_Height, 0, GL_RGBA, GL_UNSIGNED_BYTE, m_LocalBuffer
    ));

    // Sets the settings for the image so that it renders correctly
    GLCall(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST));
    GLCall(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST));
    GLCall(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE));
    GLCall(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE));

    GLCall(glGenerateTextureMipmap(m_RendererID));    // Generates a mipmap for the texture

    GLCall(glBindTexture(GL_TEXTURE_2D, 0));

    if(m_LocalBuffer)
        stbi_image_free(m_LocalBuffer);    // Frees the buffer used to load the image

    Log::variable<const std::string &>("Texture Initialised", path);
}

Texture::Texture(uint16_t width, uint16_t height, unsigned char *buffer)
    : m_RendererID(0), m_LocalBuffer(buffer), m_Width(width), m_Height(height), m_BPP(0)
{
    // This will set up textures that are created during the running of the program
    glGenTextures(1, &m_RendererID);
    glBindTexture(GL_TEXTURE_2D, m_RendererID);
    glTexImage2D(
        GL_TEXTURE_2D,
        0,
        GL_RED,
        m_Width,
        m_Height,
        0,
        GL_RED,
        GL_UNSIGNED_BYTE,
        m_LocalBuffer);
    // Sets the settings for the image so that it renders correctly
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    GLCall(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR));
    GLCall(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR));

    GLCall(glGenerateTextureMipmap(m_RendererID));    // Generates a mipmap for the texture
}

Texture::~Texture()
{
    // Delete the texture
    GLCall(glDeleteTextures(1, &m_RendererID));
}

void Texture::bind(uint8_t slot) const
```

```
{
    if(slot > 31)   // Checks slot is not over the slot limit
    {
        Log::critical("Trying to bind more than 32 textures!", LOGINFO);
    }
    bufferStorage[slot] = this;   // Updates the bufferStorage

    // Binds the texture
    GLCall(glActiveTexture(GL_TEXTURE0 + slot));
    GLCall(glBindTexture(GL_TEXTURE_2D, m_RendererID));
}

void Texture::unbind() const
{
    GLCall(glBindTexture(GL_TEXTURE_2D, 0));

    // Finds where it is bound in the bufferStorage and clears it through a simple linear search
    for(uint8_t i = 0; i < 32; i++)
    {
        if(bufferStorage[i] == this)
        {
            bufferStorage[i] = nullptr;
            break;
        }
    }
}

uint8_t Texture::getBoundSlot(Texture *tex)
{
    if(!tex)
    {
        Log::warning("Nullptr");
        return 32;
    }
    // Does a simple linear search to find where a texture is bound to
    for(uint8_t i = 0; i < 32; i++)
    {
        if(!bufferStorage[i])
            continue;
        if(bufferStorage[i]->getID() == tex->getID())
            return i;
    }
    return 32;
}

void Texture::clearBufferSlots()
{
    // Sets all the slots to nullptrs
    memset(bufferStorage , NULL, sizeof(bufferStorage));
}
```

../src/rendering/glInterface/Texture.cpp

rendering/glInterface/VertexArray.h

```
#pragma once

#include "VertexBuffer.h"
#include "VertexBufferLayout.h"

// This is for compining vertex buffer, index buffer and vertex buffer layout together
class VertexArray
{
  private:
    uint32_t m_RendererID;

  public:
    VertexArray();
    ~VertexArray();

    void addBuffer(const VertexBuffer &vb, const VertexBufferLayout &layout);

    void bind() const;
    void unbind() const;
};
```

../src/rendering/glInterface/VertexArray.h

rendering/glInterface/VertexArray.cpp

```
#include "glDebug.h"
```

```
#include "VertexArray.h"

#include "Log.h"

VertexArray::VertexArray()
{
    GLCall(glGenVertexArrays(1, &m_RendererID));
}
VertexArray::~VertexArray()
{
    GLCall(glDeleteVertexArrays(1, &m_RendererID));
}

void VertexArray::addBuffer(const VertexBuffer &vb, const VertexBufferLayout &layout)
{
    // This sets the layout of the buffer, with the given VertexBufferLayout
    bind();
    vb.bind();
    const auto& elements = layout.getElements();
    for(uint32_t i = 0; i < elements.size(); i++)
    {
        // Goes through each element and records the position of it and the type
        const auto &element = elements[i];
        GLCall(glEnableVertexAttribArray(i));
        GLCall(glVertexAttribPointer(i, element.getComponentCount(), shaderDataTypeToOpenGLBaseType(element.type),
     element.normalized, layout.getStride(), (const void *) element.offset));
    }
    vb.unbind();
}

void VertexArray::bind() const
{
    GLCall(glBindVertexArray(m_RendererID));
}
void VertexArray::unbind() const
{
    GLCall(glBindVertexArray(0));
}
```

../src/rendering/glInterface/VertexArray.cpp

rendering/glInterface/VertexBuffer.h

```
#pragma once

// This is for storing and interacting with the buffer
class VertexBuffer
{
  private:
    uint32_t     m_RendererID;
    bool         isDynamic;
    int          m_Offset;
    uint32_t     m_BufferSize;

  public:
    VertexBuffer(const void *data, uint32_t size);
    ~VertexBuffer();

    void bind() const;
    void unbind() const;
    void clearBufferData();
    bool addToBuffer(const void *vertices, uint32_t size);

    uint32_t getBufferSize() { return m_BufferSize; }
    bool     canStore(uint32_t size) { return m_Offset + size <= m_BufferSize; }
    bool     isEmpty() { return m_Offset == 0; }
};
```

../src/rendering/glInterface/VertexBuffer.h

rendering/glInterface/VertexBuffer.cpp

```
#include "glDebug.h"

#include "VertexBuffer.h"

#include "Log.h"
#include "Renderer.h"

VertexBuffer::VertexBuffer(const void *data, uint32_t size)
    : m_Offset(0), m_BufferSize(size)
{
    GLCall(glGenBuffers(1, &m_RendererID));
    GLCall(glBindBuffer(GL_ARRAY_BUFFER, m_RendererID));
    if(data)
        Log::error("Tried to create a static buffer!", LOGINFO);
    else
        GLCall(glBufferData(GL_ARRAY_BUFFER, size, data, GL_DYNAMIC_DRAW));
}

VertexBuffer::~VertexBuffer()
{
    GLCall(glDeleteBuffers(1, &m_RendererID));
}

void VertexBuffer::bind() const
{
    GLCall(glBindBuffer(GL_ARRAY_BUFFER, m_RendererID));
}

void VertexBuffer::unbind() const
{
    GLCall(glBindBuffer(GL_ARRAY_BUFFER, 0));
}

void VertexBuffer::clearBufferData()
{
    GLCall(glClearNamedBufferData(m_RendererID, GL_RGBA16, GL_RGBA, GL_UNSIGNED_BYTE, nullptr););
    m_Offset = 0;   // Resets the offset
}

bool VertexBuffer::addToBuffer(const void *vertices, uint32_t size)
{
    // Checks if the buffer is full and if it is it reports the error
    if(!canStore(size))
    {
        Log::error("Render buffer full!", LOGINFO);
        return false;
    }
    else
    {
        // Adds to the vertex buffer
        GLCall(glNamedBufferSubData(m_RendererID, m_Offset, size, vertices));

        m_Offset += size;   // Adds to the m_Offset so it correctly positions the next vertices
        return true;
    }
}
```

../src/rendering/glInterface/VertexBuffer.cpp

rendering/glInterface/VertexBufferLayout.h

```
#pragma once

#include "glDebug.h"
#include <vector>

#include "Log.h"

// These are for differentiating between the different types
enum class ShaderDataType
{
    None = 0,
    Float,
    Float2,
    Float3,
    Float4,
    Mat3,
    Mat4,
    Int,
    Int2,
    Int3,
    Int4,
    Bool
};

// This will convert each type to the corresponding openGL type
static GLenum shaderDataTypeToOpenGLBaseType(ShaderDataType type)
{
    switch(type)
    {
    case ShaderDataType::Float:
        return GL_FLOAT;
    case ShaderDataType::Float2:
        return GL_FLOAT;
    case ShaderDataType::Float3:
        return GL_FLOAT;
    case ShaderDataType::Float4:
        return GL_FLOAT;
    case ShaderDataType::Mat3:
        return GL_FLOAT;
    case ShaderDataType::Mat4:
        return GL_FLOAT;
    case ShaderDataType::Int:
        return GL_INT;
    case ShaderDataType::Int2:
        return GL_INT;
    case ShaderDataType::Int3:
        return GL_INT;
    case ShaderDataType::Int4:
        return GL_INT;
    case ShaderDataType::Bool:
        return GL_BOOL;
    }

    Log::error("Unknown ShaderDataType!", LOGINFO);
    return 0;
}

// This stores all the information needed about a single element
struct BufferElement
{
    std::string    name;
    ShaderDataType type;
    uint32_t       size;
    size_t         offset;
    bool           normalized;

    BufferElement() = default;

    BufferElement(ShaderDataType type, const std::string &name, bool normalized = false);

    uint32_t getComponentCount() const;
};

// This will store a list of buffer elements so that they can be accessed and set correctly
class VertexBufferLayout
{
  private:
    std::vector<BufferElement> m_Elements;
    uint32_t                   m_Stride = 0;

    void calculateOffsetsAndStride()
    {
        // Calculates and sets the offsets for each element
        size_t offset = 0;
        m_Stride      = 0;
        for(auto &element : m_Elements)
        {
            element.offset = offset;
            offset += element.size;
            m_Stride += element.size;
        }
    }

  public:
    VertexBufferLayout() {}

    VertexBufferLayout(std::initializer_list<BufferElement> elements)
        : m_Elements(elements)
    {
        calculateOffsetsAndStride();
    }

    uint32_t                          getStride() const { return m_Stride; }
    const std::vector<BufferElement> &getElements() const { return m_Elements; }

    // Function to allow interaction with the vector without using 'getElements'
    std::vector<BufferElement>::iterator       begin() { return m_Elements.begin(); }
    std::vector<BufferElement>::iterator       end() { return m_Elements.end(); }
    std::vector<BufferElement>::const_iterator begin() const { return m_Elements.begin(); }
    std::vector<BufferElement>::const_iterator end() const { return m_Elements.end(); }
```

```
};
```

../src/rendering/glInterface/VertexBufferLayout.h

rendering/glInterface/VertexBufferLayout.cpp

```
#include "VertexBufferLayout.h"

// This will return the size (in bytes) used by each type
static uint32_t shaderDataTypeSize(ShaderDataType type)
{
    switch(type)
    {
    case ShaderDataType::Float:
        return 4;
    case ShaderDataType::Float2:
        return 4 * 2;
    case ShaderDataType::Float3:
        return 4 * 3;
    case ShaderDataType::Float4:
        return 4 * 4;
    case ShaderDataType::Mat3:
        return 4 * 3 * 3;
    case ShaderDataType::Mat4:
        return 4 * 4 * 4;
    case ShaderDataType::Int:
        return 4;
    case ShaderDataType::Int2:
        return 4 * 2;
    case ShaderDataType::Int3:
        return 4 * 3;
    case ShaderDataType::Int4:
        return 4 * 4;
    case ShaderDataType::Bool:
        return 1;
    }

    Log::error("Unknown ShaderDataType!", LOGINFO);

    return 0;
}

BufferElement::BufferElement(ShaderDataType type, const std::string &name, bool normalized)
    : name(name), type(type), size(shaderDataTypeSize(type)), offset(0), normalized(normalized)
{
}
uint32_t BufferElement::getComponentCount() const
{
    switch(type)
    {
    case ShaderDataType::Float:
        return 1;
    case ShaderDataType::Float2:
        return 2;
    case ShaderDataType::Float3:
        return 3;
    case ShaderDataType::Float4:
        return 4;
    case ShaderDataType::Mat3:
        return 3;
    case ShaderDataType::Mat4:
        return 4;
    case ShaderDataType::Int:
        return 1;
    case ShaderDataType::Int2:
        return 2;
    case ShaderDataType::Int3:
        return 3;
    case ShaderDataType::Int4:
        return 4;
    case ShaderDataType::Bool:
        return 1;
    }

    Log::error("Unknown ShaderDataType!", LOGINFO);
    return 0;
}
```

../src/rendering/glInterface/VertexBufferLayout.cpp

3.4.3 Sprites

rendering/sprites/AnimatedSprite.h

```
#pragma once

#include "Sprite.h"

#include <vector>

class AnimatedSprite
{
  private:
    std::vector<Sprite::ID> sprites;
    int              index;
    uint16_t         textureSwapDelay, textureSwapCount;

  public:
    AnimatedSprite();
    AnimatedSprite(uint16_t frames, Sprite::ID spriteID);
    AnimatedSprite(uint16_t frames, Sprite::ID spriteID, uint16_t textureSwapDelay);

    void addSprite(Sprite::ID sprite);

    void update();

    void nextFrame();

    void setFrame(int i);

    void render(float x, float y, double rotation, float size, uint8_t layer);
    void render(float x, float y, double rotation, float width, float height, uint8_t layer);
};
```

../src/rendering/sprites/AnimatedSprite.h

rendering/sprites/AnimatedSprite.cpp

```
#include "AnimatedSprite.h"

#include "Renderer.h"

AnimatedSprite::AnimatedSprite()
    : sprites(), index(-1), textureSwapDelay(10), textureSwapCount(0)
{
}
AnimatedSprite::AnimatedSprite(uint16_t frames, Sprite::ID spriteID)
    : sprites(), index(0), textureSwapDelay(10), textureSwapCount(0)
{
    sprites.reserve(2 * frames);
    for(int i = 1; i <= frames; i++)
    {
        sprites.push_back(spriteID + i);
        sprites.push_back(spriteID);
    }
}
AnimatedSprite::AnimatedSprite(uint16_t frames, Sprite::ID spriteID, uint16_t textureSwapDelay)
    : index(0), textureSwapDelay(textureSwapDelay), textureSwapCount(0)
{
    sprites.reserve(2 * frames);
    for(int i = 1; i <= frames; i++)
    {
        sprites.push_back(spriteID + i);
        sprites.push_back(spriteID);
    }
}

void AnimatedSprite::addSprite(Sprite::ID sprite)
{
    sprites.push_back(sprite);
    if(index == -1)
        index = 0;
}

void AnimatedSprite::update()
{
    if(textureSwapCount == textureSwapDelay)
    {
        nextFrame();
        textureSwapCount = 0;
    }
    textureSwapCount++;
}

void AnimatedSprite::nextFrame()
{
    if(index != -1)
    {
        index++;
        if(index == sprites.size())
            index = 0;
    }
}

void AnimatedSprite::setFrame(int i)
{
    if(index != -1 && i > -1 && i < sprites.size())
        index = i;
}

void AnimatedSprite::render(float x, float y, double rotation, float size, uint8_t layer)
{
    // Sprite::getSprite(sprites[index])->render(x, y, rotation, size, size);
    Render::sprite(x, y, rotation, size, sprites[index], layer);
}

void AnimatedSprite::render(float x, float y, double rotation, float width, float height, uint8_t layer)
{
    // Sprite::getSprite(sprites[index])->render(x, y, rotation, width, height);
    Render::sprite(x, y, rotation, width, height, sprites[index], layer);
}
```

../src/rendering/sprites/AnimatedSprite.cpp

rendering/sprites/Sprite.h

```
#pragma once

#include <array>
#include <memory>
#include <string>

#include "Log.h"
#include "Texture.h"
#include "Utils.h"

#define SPRITE_WALK_1 static_cast<Sprite::ID>(1)
#define SPRITE_WALK_2 static_cast<Sprite::ID>(2)

#define SPRITE_NORTH static_cast<Sprite::ID>(0)
#define SPRITE_SOUTH static_cast<Sprite::ID>(3)
#define SPRITE_EAST  static_cast<Sprite::ID>(6)
#define SPRITE_WEST  static_cast<Sprite::ID>(9)
#define SPRITE_END   static_cast<Sprite::ID>(12)

#define SPRITE_FROST static_cast<Sprite::ID>(0) * SPRITE_END
#define SPRITE_FIRE  static_cast<Sprite::ID>(1) * SPRITE_END
#define SPRITE_DARK  static_cast<Sprite::ID>(2) * SPRITE_END

#define POTION_SPRITES 4
#define POTION_REGEN    static_cast<Sprite::ID>(1)
#define POTION_MAGIC    static_cast<Sprite::ID>(2)
#define POTION_HUGE     static_cast<Sprite::ID>(3)

#define SPRITE_BOOK_NUM 6
#define SPRITE_MAGIC_BOOK_NUM 2
#define SPRITE_FOOD_NUM 15

#define SPRITE_BOOK_START        Sprite::ID::books1
#define SPRITE_MAGIC_BOOK_START Sprite::ID::magicBooks1
#define SPRITE_FOOD_START        Sprite::ID::food1

class Sprite
{
  public:
    enum ID
    {
        errorID = -1,

        tileBasicWall,
        tileBasicFloor,
        tileBasicExtCorner,
        tileBasicIntCorner,
tileBasicChest,
tileBasicTrapHidden,
tileBasicTrapExposed,
tileBasicTrapdoor,

mobPlayer,
mobPlayerNorth      = mobPlayer + SPRITE_NORTH,
mobPlayerNorthWalk1 = mobPlayer + SPRITE_NORTH + SPRITE_WALK_1,
mobPlayerNorthWalk2 = mobPlayer + SPRITE_NORTH + SPRITE_WALK_2,
mobPlayerSouth      = mobPlayer + SPRITE_SOUTH,
mobPlayerSouthWalk1 = mobPlayer + SPRITE_SOUTH + SPRITE_WALK_1,
mobPlayerSouthWalk2 = mobPlayer + SPRITE_SOUTH + SPRITE_WALK_2,
mobPlayerEast       = mobPlayer + SPRITE_EAST,
mobPlayerEastWalk1  = mobPlayer + SPRITE_EAST + SPRITE_WALK_1,
mobPlayerEastWalk2  = mobPlayer + SPRITE_EAST + SPRITE_WALK_2,
mobPlayerWest       = mobPlayer + SPRITE_WEST,
mobPlayerWestWalk1  = mobPlayer + SPRITE_WEST + SPRITE_WALK_1,
mobPlayerWestWalk2  = mobPlayer + SPRITE_WEST + SPRITE_WALK_2,

followerFrost           = mobPlayer + SPRITE_END,
followerFrostNorth      = followerFrost + SPRITE_NORTH,
followerFrostNorthWalk1 = followerFrost + SPRITE_NORTH + SPRITE_WALK_1,
followerFrostNorthWalk2 = followerFrost + SPRITE_NORTH + SPRITE_WALK_2,
followerFrostSouth      = followerFrost + SPRITE_SOUTH,
followerFrostSouthWalk1 = followerFrost + SPRITE_SOUTH + SPRITE_WALK_1,
followerFrostSouthWalk2 = followerFrost + SPRITE_SOUTH + SPRITE_WALK_2,
followerFrostEast       = followerFrost + SPRITE_EAST,
followerFrostEastWalk1  = followerFrost + SPRITE_EAST + SPRITE_WALK_1,
followerFrostEastWalk2  = followerFrost + SPRITE_EAST + SPRITE_WALK_2,
followerFrostWest       = followerFrost + SPRITE_WEST,
followerFrostWestWalk1  = followerFrost + SPRITE_WEST + SPRITE_WALK_1,
followerFrostWestWalk2  = followerFrost + SPRITE_WEST + SPRITE_WALK_2,

followerFire           = followerFrost + SPRITE_END,
followerFireNorth      = followerFire + SPRITE_NORTH,
followerFireNorthWalk1 = followerFire + SPRITE_NORTH + SPRITE_WALK_1,
followerFireNorthWalk2 = followerFire + SPRITE_NORTH + SPRITE_WALK_2,
followerFireSouth      = followerFire + SPRITE_SOUTH,
followerFireSouthWalk1 = followerFire + SPRITE_SOUTH + SPRITE_WALK_1,
followerFireSouthWalk2 = followerFire + SPRITE_SOUTH + SPRITE_WALK_2,
followerFireEast       = followerFire + SPRITE_EAST,
followerFireEastWalk1  = followerFire + SPRITE_EAST + SPRITE_WALK_1,
followerFireEastWalk2  = followerFire + SPRITE_EAST + SPRITE_WALK_2,
followerFireWest       = followerFire + SPRITE_WEST,
followerFireWestWalk1  = followerFire + SPRITE_WEST + SPRITE_WALK_1,
followerFireWestWalk2  = followerFire + SPRITE_WEST + SPRITE_WALK_2,

followerDark           = followerFire + SPRITE_END,
followerDarkNorth      = followerDark + SPRITE_NORTH,
followerDarkNorthWalk1 = followerDark + SPRITE_NORTH + SPRITE_WALK_1,
followerDarkNorthWalk2 = followerDark + SPRITE_NORTH + SPRITE_WALK_2,
followerDarkSouth      = followerDark + SPRITE_SOUTH,
followerDarkSouthWalk1 = followerDark + SPRITE_SOUTH + SPRITE_WALK_1,
followerDarkSouthWalk2 = followerDark + SPRITE_SOUTH + SPRITE_WALK_2,
followerDarkEast       = followerDark + SPRITE_EAST,
followerDarkEastWalk1  = followerDark + SPRITE_EAST + SPRITE_WALK_1,
followerDarkEastWalk2  = followerDark + SPRITE_EAST + SPRITE_WALK_2,
followerDarkWest       = followerDark + SPRITE_WEST,
followerDarkWestWalk1  = followerDark + SPRITE_WEST + SPRITE_WALK_1,
followerDarkWestWalk2  = followerDark + SPRITE_WEST + SPRITE_WALK_2,

enemyFrost           = followerDark + SPRITE_END,
enemyFrostNorth      = enemyFrost + SPRITE_NORTH,
enemyFrostNorthWalk1 = enemyFrost + SPRITE_NORTH + SPRITE_WALK_1,
enemyFrostNorthWalk2 = enemyFrost + SPRITE_NORTH + SPRITE_WALK_2,
enemyFrostSouth      = enemyFrost + SPRITE_SOUTH,
enemyFrostSouthWalk1 = enemyFrost + SPRITE_SOUTH + SPRITE_WALK_1,
enemyFrostSouthWalk2 = enemyFrost + SPRITE_SOUTH + SPRITE_WALK_2,
enemyFrostEast       = enemyFrost + SPRITE_EAST,
enemyFrostEastWalk1  = enemyFrost + SPRITE_EAST + SPRITE_WALK_1,
enemyFrostEastWalk2  = enemyFrost + SPRITE_EAST + SPRITE_WALK_2,
enemyFrostWest       = enemyFrost + SPRITE_WEST,
enemyFrostWestWalk1  = enemyFrost + SPRITE_WEST + SPRITE_WALK_1,
enemyFrostWestWalk2  = enemyFrost + SPRITE_WEST + SPRITE_WALK_2,

enemyFire           = enemyFrost + SPRITE_END,
enemyFireNorth      = enemyFire + SPRITE_NORTH,
enemyFireNorthWalk1 = enemyFire + SPRITE_NORTH + SPRITE_WALK_1,
enemyFireNorthWalk2 = enemyFire + SPRITE_NORTH + SPRITE_WALK_2,
enemyFireSouth      = enemyFire + SPRITE_SOUTH,
enemyFireSouthWalk1 = enemyFire + SPRITE_SOUTH + SPRITE_WALK_1,
enemyFireSouthWalk2 = enemyFire + SPRITE_SOUTH + SPRITE_WALK_2,
enemyFireEast       = enemyFire + SPRITE_EAST,
enemyFireEastWalk1  = enemyFire + SPRITE_EAST + SPRITE_WALK_1,
enemyFireEastWalk2  = enemyFire + SPRITE_EAST + SPRITE_WALK_2,
enemyFireWest       = enemyFire + SPRITE_WEST,
enemyFireWestWalk1  = enemyFire + SPRITE_WEST + SPRITE_WALK_1,
enemyFireWestWalk2  = enemyFire + SPRITE_WEST + SPRITE_WALK_2,

enemyDark           = enemyFire + SPRITE_END,
enemyDarkNorth      = enemyDark + SPRITE_NORTH,
enemyDarkNorthWalk1 = enemyDark + SPRITE_NORTH + SPRITE_WALK_1,
enemyDarkNorthWalk2 = enemyDark + SPRITE_NORTH + SPRITE_WALK_2,
enemyDarkSouth      = enemyDark + SPRITE_SOUTH,
enemyDarkSouthWalk1 = enemyDark + SPRITE_SOUTH + SPRITE_WALK_1,
enemyDarkSouthWalk2 = enemyDark + SPRITE_SOUTH + SPRITE_WALK_2,
enemyDarkEast       = enemyDark + SPRITE_EAST,
enemyDarkEastWalk1  = enemyDark + SPRITE_EAST + SPRITE_WALK_1,
enemyDarkEastWalk2  = enemyDark + SPRITE_EAST + SPRITE_WALK_2,
enemyDarkWest       = enemyDark + SPRITE_WEST,
enemyDarkWestWalk1  = enemyDark + SPRITE_WEST + SPRITE_WALK_1,
enemyDarkWestWalk2  = enemyDark + SPRITE_WEST + SPRITE_WALK_2,

itemStick,

weaponFireStaff,
weaponFrostStaff,
weaponDarkStaff,
weaponGoldStaff,
weaponEarthStaff,
weaponAirStaff,
weaponSling,
weaponBow,
weaponCrossbow,
weaponBoomerang,

bombRed,
bombPink,
bombOrange,

potionRed,
potionRedRegen = potionRed + POTION_REGEN,
potionRedMagic = potionRed + POTION_MAGIC,
potionRedHuge  = potionRed + POTION_HUGE,

potionBlue    = potionRed + POTION_SPRITES,
potionBlueRegen = potionBlue + POTION_REGEN,
potionBlueMagic = potionBlue + POTION_MAGIC,
potionBlueHuge  = potionBlue + POTION_HUGE,

potionGreen   = potionBlue + POTION_SPRITES,
potionGreenRegen = potionGreen + POTION_REGEN,
potionGreenMagic = potionGreen + POTION_MAGIC,
potionGreenHuge  = potionGreen + POTION_HUGE,

potionYellow = potionGreen + POTION_SPRITES,
potionYellowRegen = potionYellow + POTION_REGEN,
potionYellowMagic = potionYellow + POTION_MAGIC,
potionYellowHuge  = potionYellow + POTION_HUGE,

books1,
books2,
books3,
books4,
books5,
books6,
magicBooks1,
magicBooks2,

food1,
food2,
food3,
food4,
food5,
food6,
food7,
food8,
food9,
food10,
food11,
        food12,
        food13,
        food14,
        food15,

        projectileFire,
        projectileFrost,
        projectileDark,
        projectileGold,
        projectileNature,
        projectileRock,
        projectileArrow,

        debugCircle,

        menuTitle,

        numOfSprites,


    };

  public:
    Sprite(ID id);
    Sprite(const char *texturePath);
    ~Sprite();

    void render(float x, float y, double rotation, float width, float height);

    Texture *getTexture() { return m_Texture.get(); }

    void bind(uint8_t slot = 0);
    void unbind();

    static Sprite *getSprite(ID i) { return &*sprites[static_cast<int>(i)]; };
    static void    init();

  private:
    std::shared_ptr<Texture>                                             m_Texture;
    static std::array<std::unique_ptr<Sprite>, Sprite::ID::numOfSprites> sprites;
};


inline Sprite::ID operator+(const Sprite::ID &m, const int &o)
{
    if(o < -1 || o > Sprite::ID::numOfSprites)
        Log::critical("Cannot turn into sprite!", LOGINFO);

    return static_cast<Sprite::ID>(static_cast<int>(m) + o);
}

inline Sprite::ID operator-(const Sprite::ID &m, const int &o)
{
    if(o < -1 || o > Sprite::ID::numOfSprites)
        Log::critical("Cannot turn into sprite!", LOGINFO);
    return static_cast<Sprite::ID>(static_cast<int>(m) - o);
}

inline Sprite::ID &operator+=(Sprite::ID &m, const int &o)
{
    m = m + static_cast<Sprite::ID>(o);
    return m;
}

inline Sprite::ID &operator-=(Sprite::ID &m, const int &o)
{
    m = m - static_cast<Sprite::ID>(o);
    return m;
}

inline Sprite::ID &operator++(Sprite::ID &m)
{
    m = m + static_cast<Sprite::ID>(1);
    return m;
}
```

../src/rendering/sprites/Sprite.h

rendering/sprites/Sprite.cpp

```
#include "Sprite.h"
```

```cpp
#include "SpritePaths.h"

#include <string>

std::array<std::unique_ptr<Sprite>, Sprite::ID::numOfSprites> Sprite::sprites;

Sprite::Sprite(ID id)
{
    m_Texture = std::make_unique<Texture>(getPath(id).c_str());
}

Sprite::Sprite(const char *texturePath)
{
    m_Texture = std::make_unique<Texture>(texturePath);
}

Sprite::~Sprite()
{
}

void Sprite::bind(uint8_t slot)
{
    m_Texture->bind(slot);
}

void Sprite::unbind()
{
    m_Texture->unbind();
}

void Sprite::init()
{
    for(ID id = ID::tileBasicWall; id < ID::numOfSprites; ++id)
    {
        sprites[static_cast<int>(id)] = std::make_unique<Sprite>(id);
    }

    Log::info("Sprites have been loaded");
}
```

../src/rendering/sprites/Sprite.cpp

rendering/sprites/SpritePaths.h

```cpp
#pragma once

#include "Sprite.h"

inline std::string getPath(Sprite::ID id)
{
    std::string path;
    switch(id)
    {
    case Sprite::ID::tileBasicWall:
        path = "res/textures/tiles/Wall.png";
        break;
    case Sprite::ID::tileBasicFloor:
        path = "res/textures/tiles/Floor.png";
        break;
    case Sprite::ID::tileBasicExtCorner:
        path = "res/textures/tiles/ExternalCorner.png";
        break;
    case Sprite::ID::tileBasicIntCorner:
        path = "res/textures/tiles/InternalCorner.png";
        break;
    case Sprite::ID::tileBasicChest:
        path = "res/textures/tiles/Chest.png";
        break;
    case Sprite::ID::tileBasicTrapHidden:
        path = "res/textures/tiles/BasicTrap.png";
        break;
    case Sprite::ID::tileBasicTrapExposed:
        path = "res/textures/tiles/BasicTrapExposed.png";
        break;
    case Sprite::ID::tileBasicTrapdoor:
        path = "res/textures/tiles/Trapdoor.png";
        break;
    case Sprite::ID::mobPlayerNorth:
        path = "res/textures/entities/player/heir/North.png";
        break;
    case Sprite::ID::mobPlayerNorthWalk1:
        path = "res/textures/entities/player/heir/North-Walk-1.png";
        break;
    case Sprite::ID::mobPlayerNorthWalk2:
        path = "res/textures/entities/player/heir/North-Walk-2.png";
        break;
    case Sprite::ID::mobPlayerSouth:
        path = "res/textures/entities/player/heir/South.png";
        break;
    case Sprite::ID::mobPlayerSouthWalk1:
        path = "res/textures/entities/player/heir/South-Walk-1.png";
        break;
    case Sprite::ID::mobPlayerSouthWalk2:
        path = "res/textures/entities/player/heir/South-Walk-2.png";
        break;
    case Sprite::ID::mobPlayerEast:
        path = "res/textures/entities/player/heir/East.png";
        break;
    case Sprite::ID::mobPlayerEastWalk1:
        path = "res/textures/entities/player/heir/East-Walk-1.png";
        break;
    case Sprite::ID::mobPlayerEastWalk2:
        path = "res/textures/entities/player/heir/East-Walk-2.png";
        break;
    case Sprite::ID::mobPlayerWest:
        path = "res/textures/entities/player/heir/West.png";
        break;
    case Sprite::ID::mobPlayerWestWalk1:
        path = "res/textures/entities/player/heir/West-Walk-1.png";
        break;
    case Sprite::ID::mobPlayerWestWalk2:
        path = "res/textures/entities/player/heir/West-Walk-2.png";
        break;
    case Sprite::ID::followerFrostNorth:
        path = "res/textures/entities/followers/frost/North.png";
        break;
    case Sprite::ID::followerFrostNorthWalk1:
        path = "res/textures/entities/followers/frost/North-Walk-1.png";
        break;
    case Sprite::ID::followerFrostNorthWalk2:
        path = "res/textures/entities/followers/frost/North-Walk-2.png";
        break;
    case Sprite::ID::followerFrostSouth:
        path = "res/textures/entities/followers/frost/South.png";
        break;
    case Sprite::ID::followerFrostSouthWalk1:
        path = "res/textures/entities/followers/frost/South-Walk-1.png";
        break;
    case Sprite::ID::followerFrostSouthWalk2:
        path = "res/textures/entities/followers/frost/South-Walk-2.png";
        break;
    case Sprite::ID::followerFrostEast:
        path = "res/textures/entities/followers/frost/East.png";
        break;
    case Sprite::ID::followerFrostEastWalk1:
        path = "res/textures/entities/followers/frost/East-Walk-1.png";
        break;
    case Sprite::ID::followerFrostEastWalk2:
        path = "res/textures/entities/followers/frost/East-Walk-2.png";
        break;
    case Sprite::ID::followerFrostWest:
        path = "res/textures/entities/followers/frost/West.png";
        break;
    case Sprite::ID::followerFrostWestWalk1:
        path = "res/textures/entities/followers/frost/West-Walk-1.png";
        break;
    case Sprite::ID::followerFrostWestWalk2:
        path = "res/textures/entities/followers/frost/West-Walk-2.png";
        break;
    case Sprite::ID::followerFireNorth:
        path = "res/textures/entities/followers/fire/North.png";
        break;
    case Sprite::ID::followerFireNorthWalk1:
        path = "res/textures/entities/followers/fire/North-Walk-1.png";
        break;
    case Sprite::ID::followerFireNorthWalk2:
        path = "res/textures/entities/followers/fire/North-Walk-2.png";
        break;
    case Sprite::ID::followerFireSouth:
        path = "res/textures/entities/followers/fire/South.png";
        break;
    case Sprite::ID::followerFireSouthWalk1:
        path = "res/textures/entities/followers/fire/South-Walk-1.png";
        break;
    case Sprite::ID::followerFireSouthWalk2:
        path = "res/textures/entities/followers/fire/South-Walk-2.png";
        break;
    case Sprite::ID::followerFireEast:
        path = "res/textures/entities/followers/fire/East.png";
        break;
    case Sprite::ID::followerFireEastWalk1:
        path = "res/textures/entities/followers/fire/East-Walk-1.png";
        break;
    case Sprite::ID::followerFireEastWalk2:
        path = "res/textures/entities/followers/fire/East-Walk-2.png";
        break;
    case Sprite::ID::followerFireWest:
        path = "res/textures/entities/followers/fire/West.png";
        break;
    case Sprite::ID::followerFireWestWalk1:
        path = "res/textures/entities/followers/fire/West-Walk-1.png";
        break;
    case Sprite::ID::followerFireWestWalk2:
        path = "res/textures/entities/followers/fire/West-Walk-2.png";
        break;
    case Sprite::ID::followerDarkNorth:
        path = "res/textures/entities/followers/dark/North.png";
        break;
    case Sprite::ID::followerDarkNorthWalk1:
        path = "res/textures/entities/followers/dark/North-Walk-1.png";
        break;
    case Sprite::ID::followerDarkNorthWalk2:
        path = "res/textures/entities/followers/dark/North-Walk-2.png";
        break;
    case Sprite::ID::followerDarkSouth:
        path = "res/textures/entities/followers/dark/South.png";
        break;
    case Sprite::ID::followerDarkSouthWalk1:
        path = "res/textures/entities/followers/dark/South-Walk-1.png";
        break;
    case Sprite::ID::followerDarkSouthWalk2:
        path = "res/textures/entities/followers/dark/South-Walk-2.png";
        break;
    case Sprite::ID::followerDarkEast:
        path = "res/textures/entities/followers/dark/East.png";
        break;
    case Sprite::ID::followerDarkEastWalk1:
        path = "res/textures/entities/followers/dark/East-Walk-1.png";
        break;
    case Sprite::ID::followerDarkEastWalk2:
        path = "res/textures/entities/followers/dark/East-Walk-2.png";
        break;
    case Sprite::ID::followerDarkWest:
        path = "res/textures/entities/followers/dark/West.png";
        break;
    case Sprite::ID::followerDarkWestWalk1:
        path = "res/textures/entities/followers/dark/West-Walk-1.png";
        break;
    case Sprite::ID::followerDarkWestWalk2:
        path = "res/textures/entities/followers/dark/West-Walk-2.png";
        break;
    case Sprite::ID::enemyFrostNorth:
        path = "res/textures/entities/enemies/frost/North.png";
        break;
    case Sprite::ID::enemyFrostNorthWalk1:
        path = "res/textures/entities/enemies/frost/North-Walk-1.png";
        break;
    case Sprite::ID::enemyFrostNorthWalk2:
        path = "res/textures/entities/enemies/frost/North-Walk-2.png";
        break;
    case Sprite::ID::enemyFrostSouth:
        path = "res/textures/entities/enemies/frost/South.png";
        break;
    case Sprite::ID::enemyFrostSouthWalk1:
        path = "res/textures/entities/enemies/frost/South-Walk-1.png";
        break;
    case Sprite::ID::enemyFrostSouthWalk2:
        path = "res/textures/entities/enemies/frost/South-Walk-2.png";
        break;
    case Sprite::ID::enemyFrostEast:
            path = "res/textures/entities/enemies/frost/East.png";
            break;
        case Sprite::ID::enemyFrostEastWalk1:
            path = "res/textures/entities/enemies/frost/East-Walk-1.png";
            break;
        case Sprite::ID::enemyFrostEastWalk2:
            path = "res/textures/entities/enemies/frost/East-Walk-2.png";
            break;
        case Sprite::ID::enemyFrostWest:
            path = "res/textures/entities/enemies/frost/West.png";
            break;
        case Sprite::ID::enemyFrostWestWalk1:
            path = "res/textures/entities/enemies/frost/West-Walk-1.png";
            break;
        case Sprite::ID::enemyFrostWestWalk2:
            path = "res/textures/entities/enemies/frost/West-Walk-2.png";
            break;
        case Sprite::ID::enemyFireNorth:
            path = "res/textures/entities/enemies/fire/North.png";
            break;
        case Sprite::ID::enemyFireNorthWalk1:
            path = "res/textures/entities/enemies/fire/North-Walk-1.png";
            break;
        case Sprite::ID::enemyFireNorthWalk2:
            path = "res/textures/entities/enemies/fire/North-Walk-2.png";
            break;
        case Sprite::ID::enemyFireSouth:
            path = "res/textures/entities/enemies/fire/South.png";
            break;
        case Sprite::ID::enemyFireSouthWalk1:
            path = "res/textures/entities/enemies/fire/South-Walk-1.png";
            break;
        case Sprite::ID::enemyFireSouthWalk2:
            path = "res/textures/entities/enemies/fire/South-Walk-2.png";
            break;
        case Sprite::ID::enemyFireEast:
            path = "res/textures/entities/enemies/fire/East.png";
            break;
        case Sprite::ID::enemyFireEastWalk1:
            path = "res/textures/entities/enemies/fire/East-Walk-1.png";
            break;
        case Sprite::ID::enemyFireEastWalk2:
            path = "res/textures/entities/enemies/fire/East-Walk-2.png";
            break;
        case Sprite::ID::enemyFireWest:
            path = "res/textures/entities/enemies/fire/West.png";
            break;
        case Sprite::ID::enemyFireWestWalk1:
            path = "res/textures/entities/enemies/fire/West-Walk-1.png";
            break;
        case Sprite::ID::enemyFireWestWalk2:
            path = "res/textures/entities/enemies/fire/West-Walk-2.png";
            break;
        case Sprite::ID::enemyDarkNorth:
            path = "res/textures/entities/enemies/dark/North.png";
            break;
        case Sprite::ID::enemyDarkNorthWalk1:
            path = "res/textures/entities/enemies/dark/North-Walk-1.png";
            break;
        case Sprite::ID::enemyDarkNorthWalk2:
            path = "res/textures/entities/enemies/dark/North-Walk-2.png";
            break;
        case Sprite::ID::enemyDarkSouth:
            path = "res/textures/entities/enemies/dark/South.png";
            break;
        case Sprite::ID::enemyDarkSouthWalk1:
            path = "res/textures/entities/enemies/dark/South-Walk-1.png";
            break;
        case Sprite::ID::enemyDarkSouthWalk2:
            path = "res/textures/entities/enemies/dark/South-Walk-2.png";
            break;
        case Sprite::ID::enemyDarkEast:
            path = "res/textures/entities/enemies/dark/East.png";
            break;
        case Sprite::ID::enemyDarkEastWalk1:
            path = "res/textures/entities/enemies/dark/East-Walk-1.png";
            break;
        case Sprite::ID::enemyDarkEastWalk2:
            path = "res/textures/entities/enemies/dark/East-Walk-2.png";
            break;
```

```
    case Sprite::ID::enemyDarkWest:
        path = "res/textures/entities/enemies/dark/West.png";
        break;
    case Sprite::ID::enemyDarkWestWalk1:
        path = "res/textures/entities/enemies/dark/West-Walk-1.png";
        break;
    case Sprite::ID::enemyDarkWestWalk2:
        path = "res/textures/entities/enemies/dark/West-Walk-2.png";
        break;
    case Sprite::ID::itemStick:
        path = "res/textures/items/Stick.png";
        break;
    case Sprite::ID::weaponFireStaff:
        path = "res/textures/items/weapons/FireStaff.png";
        break;
    case Sprite::ID::weaponFrostStaff:
        path = "res/textures/items/weapons/FrostStaff.png";
        break;
    case Sprite::ID::weaponDarkStaff:
        path = "res/textures/items/weapons/DarkStaff.png";
        break;
    case Sprite::ID::weaponGoldStaff:
        path = "res/textures/items/weapons/GoldStaff.png";
        break;
    case Sprite::ID::weaponEarthStaff:
        path = "res/textures/items/weapons/EarthStaff.png";
        break;
    case Sprite::ID::weaponAirStaff:
        path = "res/textures/items/weapons/AirStaff.png";
        break;
    case Sprite::ID::weaponSling:
        path = "res/textures/items/weapons/Sling.png";
        break;
    case Sprite::ID::weaponBow:
        path = "res/textures/items/weapons/Bow.png";
        break;
    case Sprite::ID::weaponCrossbow:
        path = "res/textures/items/weapons/Crossbow.png";
        break;
    case Sprite::ID::weaponBoomerang:
        path = "res/textures/items/weapons/Boomerang.png";
        break;
    case Sprite::ID::bombRed:
        path = "res/textures/items/weapons/RedBomb.png";
        break;
    case Sprite::ID::bombPink:
        path = "res/textures/items/weapons/PinkBomb.png";
        break;
    case Sprite::ID::bombOrange:
        path = "res/textures/items/weapons/OrangeBomb.png";
        break;
    case Sprite::ID::potionRed:
        path = "res/textures/items/potions/bottles/Red.png";
        break;
    case Sprite::ID::potionRedRegen:
        path = "res/textures/items/potions/bottles/RedRegen.png";
        break;
    case Sprite::ID::potionRedMagic:
        path = "res/textures/items/potions/bottles/RedMagic.png";
        break;
    case Sprite::ID::potionRedHuge:
        path = "res/textures/items/potions/bottles/RedHuge.png";
        break;
    case Sprite::ID::potionBlue:
        path = "res/textures/items/potions/bottles/Blue.png";
        break;
    case Sprite::ID::potionBlueRegen:
        path = "res/textures/items/potions/bottles/BlueRegen.png";
        break;
    case Sprite::ID::potionBlueMagic:
        path = "res/textures/items/potions/bottles/BlueMagic.png";
        break;
    case Sprite::ID::potionBlueHuge:
        path = "res/textures/items/potions/bottles/BlueHuge.png";
        break;
    case Sprite::ID::potionGreen:
        path = "res/textures/items/potions/bottles/Green.png";
        break;
    case Sprite::ID::potionGreenRegen:
        path = "res/textures/items/potions/bottles/GreenRegen.png";
        break;
    case Sprite::ID::potionGreenMagic:
        path = "res/textures/items/potions/bottles/GreenMagic.png";
        break;
    case Sprite::ID::potionGreenHuge:
        path = "res/textures/items/potions/bottles/GreenHuge.png";
        break;
    case Sprite::ID::potionYellow:
        path = "res/textures/items/potions/bottles/Yellow.png";
        break;
    case Sprite::ID::potionYellowRegen:
        path = "res/textures/items/potions/bottles/YellowRegen.png";
        break;
    case Sprite::ID::potionYellowMagic:
        path = "res/textures/items/potions/bottles/YellowMagic.png";
        break;
    case Sprite::ID::potionYellowHuge:
        path = "res/textures/items/potions/bottles/YellowHuge.png";
        break;
    case Sprite::ID::books1:
        path = "res/textures/items/potions/books/Blue.png";
        break;
    case Sprite::ID::books2:
        path = "res/textures/items/potions/books/DarkBrown.png";
        break;
    case Sprite::ID::books3:
        path = "res/textures/items/potions/books/Green.png";
        break;
    case Sprite::ID::books4:
        path = "res/textures/items/potions/books/LightBrown.png";
        break;
    case Sprite::ID::books5:
        path = "res/textures/items/potions/books/Red.png";
        break;
    case Sprite::ID::books6:
        path = "res/textures/items/potions/books/Yellow.png";
        break;
    case Sprite::ID::magicBooks1:
        path = "res/textures/items/potions/books/MagicYellow.png";
        break;
    case Sprite::ID::magicBooks2:
        path = "res/textures/items/potions/books/MagicRed.png";
        break;
    case Sprite::ID::food1:
        path = "res/textures/items/potions/food/Apple.png";
        break;
    case Sprite::ID::food2:
        path = "res/textures/items/potions/food/Baguette.png";
        break;
    case Sprite::ID::food3:
        path = "res/textures/items/potions/food/Banana.png";
        break;
    case Sprite::ID::food4:
        path = "res/textures/items/potions/food/Boar.png";
        break;
    case Sprite::ID::food5:
        path = "res/textures/items/potions/food/Bread.png";
        break;
    case Sprite::ID::food6:
        path = "res/textures/items/potions/food/Carrot.png";
        break;
    case Sprite::ID::food7:
        path = "res/textures/items/potions/food/Chicken.png";
        break;
    case Sprite::ID::food8:
        path = "res/textures/items/potions/food/ChickenLeg.png";
        break;
    case Sprite::ID::food9:
        path = "res/textures/items/potions/food/Fish.png";
        break;
    case Sprite::ID::food10:
        path = "res/textures/items/potions/food/Grapes.png";
        break;
    case Sprite::ID::food11:
        path = "res/textures/items/potions/food/Lemon.png";
        break;
    case Sprite::ID::food12:
        path = "res/textures/items/potions/food/Meat.png";
        break;
    case Sprite::ID::food13:
        path = "res/textures/items/potions/food/Pear.png";
        break;
    case Sprite::ID::food14:
        path = "res/textures/items/potions/food/Strawberry.png";
        break;
    case Sprite::ID::food15:
        path = "res/textures/items/potions/food/Tomato.png";
        break;
    case Sprite::ID::projectileFire:
        path = "res/textures/projectiles/Fire.png";
        break;
    case Sprite::ID::projectileFrost:
        path = "res/textures/projectiles/Frost.png";
        break;
    case Sprite::ID::projectileDark:
        path = "res/textures/projectiles/Dark.png";
        break;
    case Sprite::ID::projectileGold:
        path = "res/textures/projectiles/Gold.png";
        break;
    case Sprite::ID::projectileNature:
        path = "res/textures/projectiles/Nature.png";
        break;
    case Sprite::ID::projectileRock:
        path = "res/textures/projectiles/Rock.png";
        break;
    case Sprite::ID::projectileArrow:
        path = "res/textures/projectiles/Arrow.png";
        break;
    case Sprite::ID::debugCircle:
        path = "res/textures/DebugCircle.png";
        break;
    case Sprite::ID::menuTitle:
        path = "res/menus/Title.png";
        break;
    default:
        Log::warning("Unknown sprite!");
        path = "res/textures/DebugCircle.png";
        break;
    }
    return path;
}
```

../src/rendering/sprites/SpritePaths.h

3.5 Particle

3.5.1 particle

particle/particle/Particle.h

```
#pragma once

#include "Renderer.h"
#include "Utils.h"

#include "Event.h"

class Particle
{
  private:
    float x, y;
    float m_Size;
    Vec2f m_Dir;
    glm::vec4 m_Colour;

    uint16_t m_Lifetime;
    uint16_t m_Age;

  public:
    Particle(float x, float y, float size, Vec2f dir, uint16_t lifetime, glm::vec4 colour);
    ~Particle();

    virtual void render();
    virtual void update();
    virtual bool eventCallback(const Event::Event &e);

    void changeX(float changeBy);
    void changeY(float changeBy);
    float getX();
    float getY();
    float getSize();

    bool deleteMe();
};
```

../src/particle/particle/Particle.h

particle/particle/Particle.cpp

```
#include "Particle.h"

#include "Application.h"

Particle::Particle(float x, float y, float size, Vec2f dir, uint16_t lifetime, glm::vec4 colour)
    : x(x), y(y), m_Size(size), m_Dir(dir), m_Lifetime(lifetime), m_Age(0), m_Colour(colour)
{
}

Particle::~Particle()
{
}

void Particle::render()
{
    uint8_t layer = 4;
    Render::rectangle(x, y, 0.0f, m_Size, m_Size, m_Colour, layer);
}

void Particle::update()
{
    if(!deleteMe())
    {
        x += m_Dir.x;
        y += m_Dir.y;

        m_Age++;
    }
}

bool Particle::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        const Event::MazeMovedEvent &ne = static_cast<const Event::MazeMovedEvent &>(e);
        x += ne.changeX;
        y += ne.changeY;
    }
    return false;
}

void Particle::changeX(float changeBy) { x += changeBy; }
void Particle::changeY(float changeBy) { y += changeBy; }

float Particle::getX() { return x; }
float Particle::getY() { return y; }
float Particle::getSize() { return m_Size; }
bool  Particle::deleteMe() { return m_Age == m_Lifetime; }
```

../src/particle/particle/Particle.cpp

3.5.2 spawner

particle/spawner/Spawner.h

```
#pragma once

#include "Event.h"

class Level;
class Spawner
{
  protected:
    float  x, y;
    Level *m_Level;

    uint16_t m_Lifetime;
    uint16_t m_Age;
  public:
    Spawner();
    Spawner(float x, float y, Level *level);
    Spawner(float x, float y, Level *level, uint16_t lifetime);
    virtual ~Spawner();

    virtual void render() = 0;
    virtual void update();
    virtual bool eventCallback(const Event::Event &e);

    virtual bool deleteMe() { return m_Age == m_Lifetime; }

    float getX() { return x; }
    float getY() { return y; }
};
```

../src/particle/spawner/Spawner.h

particle/spawner/Spawner.cpp

```
#include "Spawner.h"

#include "Level.h"

Spawner::Spawner()
    : x(3500.0f), y(3500.0f), m_Level(nullptr), m_Lifetime(60), m_Age(0)
{
}

Spawner::Spawner(float x, float y, Level *level)
    : x(x), y(y), m_Level(level), m_Lifetime(60), m_Age(0)
{
}

Spawner::Spawner(float x, float y, Level *level, uint16_t lifetime)
    : x(x), y(y), m_Level(level), m_Lifetime(lifetime), m_Age(0)
{
}

Spawner::~Spawner()
{
}

void Spawner::update()
{
    if(!deleteMe())
        m_Age++;
}

bool Spawner::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        const Event::MazeMovedEvent &ne = static_cast<const Event::MazeMovedEvent &>(e);
        x += ne.changeX;
        y += ne.changeY;
    }
    return false;
}
```

../src/particle/spawner/Spawner.cpp

particle/spawner/ParticleSpawner.h

```
#pragma once

#include "Spawner.h"

#include <GLM.h>
#include "Particle.h"

class ParticleSpawner : public Spawner
{
  protected:
    std::vector<Particle> m_Particles;

    uint16_t m_ParticleSpawnRate;
    uint16_t m_ParticleMinLifeTime;
    uint16_t m_ParticleMaxLifeTime;

    float m_ParticleXMinSpeed;
    float m_ParticleXMaxSpeed;
    float m_ParticleYMinSpeed;
    float m_ParticleYMaxSpeed;

    float m_ParticleMinSize;
    float m_ParticleMaxSize;

    uint16_t   m_NumOfParticles;
    glm::vec4 m_Colour;

  public:
    ParticleSpawner();
    ParticleSpawner(float x, float y, Level *level);
    ParticleSpawner(float x, float y, Level *level, uint16_t mLife, uint16_t spawnRate, uint16_t minLife, uint16_t
     maxLife, float xMinSpeed, float xMaxSpeed, float yMinSpeed, float yMaxSpeed, float minSize, float maxSize,
    uint16_t groupSize, glm::vec4 colour);
    virtual ~ParticleSpawner() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;

    glm::vec4 *             getColour();
    std::vector<Particle> *getParticles();

    virtual bool deleteMe() override;
};
```

../src/particle/spawner/ParticleSpawner.h

particle/spawner/ParticleSpawner.cpp

```
#include "ParticleSpawner.h"

#include "Level.h"
#include "RandomGen.h"

ParticleSpawner::ParticleSpawner()
    : Spawner(), m_ParticleSpawnRate(4), m_Colour({0.0f, 0.0f, 0.0f, 1.0f})
{
}

ParticleSpawner::ParticleSpawner(float x, float y, Level *level)
    : Spawner(x, y, level),
      m_ParticleSpawnRate(4),
      m_ParticleMinLifeTime(5),
      m_ParticleMaxLifeTime(20),
      m_ParticleXMinSpeed(-1.0f),
      m_ParticleXMaxSpeed(1.0f),
      m_ParticleYMinSpeed(-3.0f),
      m_ParticleYMaxSpeed(3.0f),
      m_ParticleMinSize(5.0f),
      m_ParticleMaxSize(20.0f),
      m_NumOfParticles(3),
      m_Colour({0.0f, 0.0f, 0.0f, 1.0f})
{
}

ParticleSpawner::ParticleSpawner(float x, float y, Level *level, uint16_t mLife, uint16_t spawnRate, uint16_t
    minLife, uint16_t maxLife, float xMinSpeed, float xMaxSpeed, float yMinSpeed, float yMaxSpeed, float minSize,
    float maxSize, uint16_t groupSize, glm::vec4 colour)
    : Spawner(x, y, level, mLife),
      m_ParticleSpawnRate(spawnRate),
      m_ParticleMinLifeTime(minLife),
      m_ParticleMaxLifeTime(maxLife),
      m_ParticleXMinSpeed(xMinSpeed),
      m_ParticleXMaxSpeed(xMaxSpeed),
      m_ParticleYMinSpeed(yMinSpeed),
      m_ParticleYMaxSpeed(yMaxSpeed),
      m_ParticleMinSize(minSize),
      m_ParticleMaxSize(maxSize),
      m_NumOfParticles(groupSize),
      m_Colour(colour)
{
}

ParticleSpawner::~ParticleSpawner()
{
}

void ParticleSpawner::render()
```

```
{
    for(Particle p : m_Particles)
        p.render();
}

void ParticleSpawner::update()
{
    if(m_Age != m_Lifetime)
    {
        if(m_Age % m_ParticleSpawnRate == 0)
        {
            for(int i = 0; i < m_NumOfParticles; i++)
            {
                int      xGen     = Random::getNum((int) (m_ParticleXMinSpeed * 10), (int) (m_ParticleXMaxSpeed *
    10));
                int      yGen     = Random::getNum((int) (m_ParticleYMinSpeed * 10), (int) (m_ParticleYMaxSpeed *
    10));
                Vec2f    dir      = {(float) xGen / 10, (float) yGen / 10};
                float    size     = (float) Random::getNum((int) (m_ParticleMinSize * 10), (int) (
    m_ParticleMaxSize * 10)) / 10;
                uint16_t lifetime = Random::getNum(m_ParticleMinLifeTime, m_ParticleMaxLifeTime);
                m_Particles.emplace_back(x, y, size, dir, lifetime, m_Colour);
            }
        }
        m_Age++;
    }

    for(auto it = m_Particles.begin(); it != m_Particles.end();)
    {
        (*it).update();
        if((*it).deleteMe())
            it = m_Particles.erase(it);
        else
            ++it;
    }
}

bool ParticleSpawner::eventCallback(const Event::Event &e)
{
    for(Particle p : m_Particles)
    {
        if(p.eventCallback(e))
            return true;
    }
    return Spawner::eventCallback(e);
}

glm::vec4 *ParticleSpawner::getColour()
{
    return &m_Colour;
}

std::vector<Particle> *ParticleSpawner::getParticles()
{
    return &m_Particles;
}

bool ParticleSpawner::deleteMe()
{
    return m_Age == m_Lifetime && m_Particles.size() == 0;
}
```

../src/particle/spawner/ParticleSpawner.cpp

3.6 Maze

3.6.1 Room

maze/rooms/Room.h

```
#pragma once

#include "Camera.h"
#include "Entity.h"
#include "Event.h"
#include "KeyDefinitions.h"
#include "Renderer.h"
#include "Tile.h"
#include "Utils.h"
```

```cpp
#include <array>
#include <vector>

class Level;

class Room
{
  protected:
    float               x, y;
    bool                m_Entrances[4];    // 0: North 1: South 2: East 3: West
    bool                 isLocked;
    RoomType             m_Type;
    Level *              m_Level;

    std::vector<Entity *> m_Entities;

    std::array<Tile *, ROOM_SIZE * ROOM_SIZE> m_Tiles;    // NOTE: Please do not store this class on the stack!

  public:
    Room(float x, float y, bool entrances[4], RoomType type, Level *level);
    virtual ~Room();
    virtual void render();
    virtual void update();
    virtual bool eventCallback(const Event::Event &e);

#ifdef DEBUG
    virtual void imGuiRender();
#endif

    Entity *entityCollisionDetection(float nextX, float nextY, CollisionBox box);

    virtual bool isOpen(Direction entrance);

    void addEntity(Entity *e) { m_Entities.push_back(e); }

    virtual void active();

    Tile *getTile(int x, int y);

    void checkForMobs();
};
```

../src/maze/rooms/Room.h

maze/rooms/Room.cpp

```cpp
#include "Room.h"
#include "RoomColours.h"

#include "Application.h"
#include "Level.h"
#include "Log.h"
#include "RandomGen.h"
#include "Sprite.h"

#include "NPC.h"

#include "Chest.h"
#include "SwitchTile.h"
#include "Trap.h"
#include "Trapdoor.h"

#include <algorithm>
#include <stb_image/stb_image.h>

#define _USE_MATH_DEFINES
#ifdef IS_ON_WINDOWS
    #include <math.h>
#else
    #include <cmath>
#endif

#define ROOMS_FOLDER "res/rooms/"

Room::Room(float x, float y, bool entrances[4], RoomType type, Level *level)
    : x(x), y(y), isLocked(false), m_Type(type), m_Level(level)
{
    for(int i = 0; i < 4; i++)
        m_Entrances[i] = entrances[i];
```

```
std::string filePath = ROOMS_FOLDER;
if(type == RoomType::Chest)
    filePath.append("Chest.png");
else if(type == RoomType::Trap)
    filePath.append("Trap.png");
else if(type == RoomType::Exit)
    filePath.append("Exit.png");
else
    filePath.append("Empty.png");

if(type == RoomType::Enemy)
{
    NPC *enemy = new NPC(x + (ROOM_SIZE / 2) * TILE_SIZE, y + (ROOM_SIZE / 2) * TILE_SIZE, m_Level, NPC::Type::Enemy);
    enemy->setEnemy(m_Level->getPlayer());
    m_Entities.push_back(enemy);
    while(enemy->canAddFollower())
    {
        int r = Random::getNum(0, 4);
        if(r = 0)
        {
            NPC *follower = new NPC(x + (ROOM_SIZE / 2) * TILE_SIZE, y + (ROOM_SIZE / 2) * TILE_SIZE, m_Level, NPC::Type::Follower);
            enemy->addFollower(follower);
            m_Entities.push_back(follower);
        }
        else
            break;
    }
}
else if(type == RoomType::NPC)
{
    NPC *npc = new NPC(x + (ROOM_SIZE / 2) * TILE_SIZE, y + (ROOM_SIZE / 2) * TILE_SIZE, m_Level, NPC::Type::Follower);
    m_Entities.push_back(npc);
}

int width, height, comp;
stbi_set_flip_vertically_on_load(1);   // This loads the bitmap file that contains the information to create the room
unsigned char *data = stbi_load(filePath.c_str(), &width, &height, &comp, 4);

if(height != width || height != ROOM_SIZE)
    Log::critical("Room file not configured properly!", LOGINFO);

for(int i = 0; i < height; i++)
{
    for(int j = 0; j < width; j++)
    {   // This goes through each pixel to determine what tile will be placed

        Vec2f pos = {x + j * TILE_SIZE, y + i * TILE_SIZE};

        unsigned char *pixelOffset = data + (i * width + j) * 4;
        // TODO: Have this programmed into the file or something nicer than this
        // This checks if any of the entrances are closed
        if(pixelOffset[0] == CHEST_COLOUR)
        {
            m_Tiles[i * ROOM_SIZE + j] = new Chest(pos.x, pos.y, 0.0f, m_Level, false);   // This creates the tile and adds it to the vector
        }
        else if(pixelOffset[0] == DUD_CHEST_COLOUR)
        {
            m_Tiles[i * ROOM_SIZE + j] = new Chest(pos.x, pos.y, 0.0f, m_Level, true);   // This creates the tile and adds it to the vector
        }
        else if(pixelOffset[0] == TRAP_COLOUR)
        {
            m_Tiles[i * ROOM_SIZE + j] = new Trap(pos.x, pos.y, 0.0f, m_Level);   // This creates the tile and adds it to the vector
        }
        else if(pixelOffset[0] == TRAPDOOR_COLOUR)
        {
            m_Tiles[i * ROOM_SIZE + j] = new Trapdoor(pos.x, pos.y, 0.0f, m_Level);   // This creates the tile and adds it to the vector
        }
        else
        {
            Sprite::ID  texID;
            bool    isSolid  = true;
            double rotation = 0.0f;
            bool        isSwitch = false;
            double      altRotation;

            if(!m_Entrances[Direction::north] && i == height - 1 && j != 0 && j != width - 1)
            {
                texID = Sprite::ID::tileBasicWall;
            }
            else if(!m_Entrances[Direction::south] && i == 0 && j != 0 && j != width - 1)
            {
                texID    = Sprite::ID::tileBasicWall;
                rotation = M_PI;
            }
            else if(!m_Entrances[Direction::east] && j == width - 1 && i != 0 && i != height - 1)
            {
                texID    = Sprite::ID::tileBasicWall;
                rotation = M_PI / 2;
            }
            else if(!m_Entrances[Direction::west] && j == 0 && i != 0 && i != height - 1)
            {
                texID    = Sprite::ID::tileBasicWall;
                rotation = 3 * M_PI / 2;
            }
            else
            {
                if(((j == 0 || j == width - 1) && i > height / 2 - 2 && i < height / 2 + 2) || ((i == 0 || i
== height - 1) && j > width / 2 - 2 && j < width / 2 + 2))
                {
                    if(j == 0)   // Makes sure that the rotation is correct
                        altRotation = 3 * M_PI / 2;
                    else if(j == width - 1)
                        altRotation = M_PI / 2;
                    else if(i == 0)
                        altRotation = M_PI;
                    else
                        altRotation = 0.0f;
                    isSwitch = true;
                }

                if(pixelOffset[0] == WALL_COLOUR)   // Checks the colour against the different defined ones
                {
                    texID = Sprite::ID::tileBasicWall;
                    if(j == 0)   // Makes sure that the rotation is correct
                        rotation = 3 * M_PI / 2;
                    else if(j == width - 1)
                        rotation = M_PI / 2;
                    else if(i == 0)
                        rotation = M_PI;
                }
                else if(pixelOffset[0] == FLOOR_COLOUR)
                {
                    isSolid = false;
                    if(i == height - 1 && !entrances[0])
                        texID = Sprite::ID::tileBasicWall;
                    else
                        texID = Sprite::ID::tileBasicFloor;
                }
                else if(pixelOffset[0] == CORNER_OUT_COLOUR)
                {
                    texID = Sprite::ID::tileBasicExtCorner;
                    if(j <= width / 2 && i <= height / 2)
                        rotation = 3 * M_PI / 2;
                    else if(j > width / 2 && i > height / 2)
                        rotation = M_PI / 2;
                    else if(j > width / 2 && i <= height / 2)
                        rotation = M_PI;
                }
                else if(pixelOffset[0] == CORNER_IN_COLOUR)
                {
                    texID = Sprite::ID::tileBasicIntCorner;
                    if(j <= width / 2 && i <= height / 2)
                        rotation = 3 * M_PI / 2;
                    else if(j > width / 2 && i > height / 2)
                        rotation = M_PI / 2;
                    else if(j > width / 2 && i <= height / 2)
                        rotation = M_PI;
                }
                else
                    continue;   // If it is an unknown colour it continues
            }
```

```
                if(isSwitch)
                {
                    m_Tiles[i * ROOM_SIZE + j] = new SwitchTile(pos.x, pos.y, rotation, texID, altRotation, Sprite::ID::tileBasicWall, isSolid, m_Level);   // This creates the tile and adds it to the vector
                }
                else
                    m_Tiles[i * ROOM_SIZE + j] = new Tile(pos.x, pos.y, rotation, texID, isSolid, m_Level);   // This creates the tile and adds it to the vector
            }
        }
    }
}
Room::~Room()
{
    for(Tile *tile : m_Tiles)
        delete tile;
    for(Entity *entity : m_Entities)
        delete entity;
}

void Room::render()
{
    for(Tile *tile : m_Tiles)
        tile->render();
    for(Entity *entity : m_Entities)
        entity->render();
}

void Room::update()
{
    for(Tile *tile : m_Tiles)
        tile->update();

    for(auto it = m_Entities.begin(); it != m_Entities.end();)
    {
        (*it)->update();
        if((*it)->deleteMe())
        {
            Mob *mob = dynamic_cast<Mob *>(*it);
            if(mob)
            {
                Application::callEventLater(new Event::MobDied(mob));
                ++it;
            }
            else
            {
                delete *it;
                it = m_Entities.erase(it);
            }
        }
        else
            ++it;
    }
}

bool Room::eventCallback(const Event::Event &e)
{
    bool moveEntity = false;

    // TODO: Make all these case switch statements
    if(e.getType() == Event::EventType::showAltTileEvent)
    {
        const Event::ShowAltTileEvent &ne = static_cast<const Event::ShowAltTileEvent &>(e);
        if(ne.showAlt)
        {
            m_Type   = RoomType::Enemy;
            isLocked = true;
        }
        else
        {
            m_Type   = RoomType::Empty;
            isLocked = false;
        }
    }
    else if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        const Event::MazeMovedEvent &ne = static_cast<const Event::MazeMovedEvent &>(e);
        x += ne.changeX;
        y += ne.changeY;
    }
```

```cpp
    else if(e.getType() == Event::EventType::playerResponse)
    {
        const Event::PlayerResponse &ne = static_cast<const Event::PlayerResponse &>(e);

        if(ne.response == Event::PlayerResponse::accept)
            moveEntity = true;
    }
    else if(e.getType() == Event::EventType::mobDied)
    {
        const Event::MobDied &ne = static_cast<const Event::MobDied &>(e);

        auto index = std::find(m_Entities.begin(), m_Entities.end(), ne.mob);
        if(index != m_Entities.end())
        {
            delete ne.mob;
            m_Entities.erase(index);
            checkForMobs();
        }
    }

    for(Tile *tile : m_Tiles)
    {
        if(tile->eventCallback(e))
            return true;
    }

    for(int i = 0; i < m_Entities.size(); i++)
    {
        if(m_Entities[i]->eventCallback(e))
        {
            if(moveEntity)
            {
                m_Level->addEntity(m_Entities[i]);

                m_Entities.erase(m_Entities.begin() + i);
            }

            return true;
        }
    }

    return false;
}

bool Room::isOpen(Direction entrance)
{
    return !isLocked && m_Entrances[entrance];
}

void Room::active()
{
    if(m_Type == RoomType::Enemy)
    {
        Event::ShowAltTileEvent e(true);
        Application::callEvent(e);
    }
}

Entity *Room::entityCollisionDetection(float nextX, float nextY, CollisionBox box)
{
    for(Entity *e : m_Entities)
    {
        if(e->hasCollidedWith(nextX, nextY, box))
            return e;
    }

    return nullptr;
}

Tile *Room::getTile(int x, int y) { return m_Tiles[y * ROOM_SIZE + x]; }

void Room::checkForMobs()
{
    if(isLocked)
    {
        auto searchFunc = [this](const Entity *o) -> bool {
            const Mob *mob = dynamic_cast<const Mob *>(o);
            return mob && mob->getEnemy() == m_Level->getPlayer();
        };
        std::vector<Entity *>::iterator it = std::find_if(m_Entities.begin(), m_Entities.end(), searchFunc);
        if(it == m_Entities.end())
        {
            Event::ShowAltTileEvent e(false);
            Application::callEvent(e);
        }
    }
}

#ifdef DEBUG

void Room::imGuiRender()
{
}

#endif
```

../src/maze/rooms/Room.cpp

3.6.2 Tiles

maze/tiles/Tile.h

```
#pragma once

#include "Renderer.h"

#include "Camera.h"
#include "KeyDefinitions.h"

class Level;   // FIXME
class Tile
{
  protected:
    Sprite::ID m_SpriteID;
    float  x, y;
    double rotation;
    bool   m_IsSolid;
    Level *m_Level;

  public:
    Tile();
    Tile(float x, float y, double rotation, Sprite::ID id, bool isSolid, Level *level);
    virtual ~Tile();

    virtual void render();
    virtual void update();
    virtual bool eventCallback(const Event::Event &e);
#ifdef DEBUG
    virtual void imGuiRender();
#endif

    virtual bool isSolid()
    {
        return m_IsSolid;
    }
};
```

../src/maze/tiles/Tile.h

maze/tiles/Tile.cpp

```
#include "Tile.h"

#include "Application.h"
#include "Level.h"
#include "Log.h"
#include "Sprite.h"

Tile::Tile()
    : m_SpriteID(Sprite::ID::errorID), x(0.0f), y(0.0f), rotation(0.0f), m_IsSolid(false), m_Level(nullptr)
{
}
Tile::Tile(float x, float y, double rotation, Sprite::ID id, bool isSolid, Level *level)
    : m_SpriteID(id), x(x), y(y), rotation(rotation), m_IsSolid(isSolid), m_Level(level)
{
}

Tile::~Tile()
{
}
```

```
void Tile::render()
{
    uint8_t layer = 0;
    Render::sprite(x, y, rotation, TILE_SIZE, m_SpriteID, layer);
}

void Tile::update()
{
}

bool Tile::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        const Event::MazeMovedEvent &ne = static_cast<const Event::MazeMovedEvent &>(e);
        x += ne.changeX;
        y += ne.changeY;
    }

    return false;
}

#ifdef DEBUG
void Tile::imGuiRender()
{
}
#endif
```

../src/maze/tiles/Tile.cpp

maze/tiles/Chest.h

```
#pragma once

#include "Button.h"
#include "Item.h"
#include "ItemContainer.h"
#include "Tile.h"
#include "Utils.h"

class Chest : public Tile
{
  protected:
    ItemContainer        m_Inventory;
    Button::State        m_State;
    bool                 m_IsDud;

    void generateInventory();

    CollisionBox getCollisionBox();

  public:
    Chest();
    Chest(float x, float y, double rotation, Level *level, bool isDud);
    virtual ~Chest() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif

    ItemContainer &getInventory()
    {
        return m_Inventory;
    }
};
```

../src/maze/tiles/Chest.h

maze/tiles/Chest.cpp

```
#include "Chest.h"

#include "Application.h"
#include "FireStaff.h"
#include "KeyDefinitions.h"
#include "Level.h"
#include "Player.h"
#include "RandomGen.h"
```

```
#include "Sprite.h"
#include "Utils.h"

Chest::Chest()
    : Tile(), m_Inventory(DEFAULT_INVENTORY_SIZE), m_State(Button::State::None)
{
}

Chest::Chest(float x, float y, double rotation, Level *level, bool isDud)
    : Tile(x, y, rotation, Sprite::ID::tileBasicChest, true, level), m_Inventory(DEFAULT_INVENTORY_SIZE), m_State(
    Button::State::None), m_IsDud(isDud)
{
    if(!m_IsDud)
        generateInventory();
}

Chest::~Chest()
{
}

void Chest::generateInventory()
{
    int numOfItems = Random::getNum(2, 10);
    for(int i = 0; i < numOfItems; i++)
        m_Inventory.push_back(Random::getItem());
}

CollisionBox Chest::getCollisionBox()
{
    return {{-TILE_SIZE / 2, -TILE_SIZE / 2}, {TILE_SIZE / 2, TILE_SIZE / 2}};
}

void Chest::render()
{
    Render::sprite(x, y, rotation, TILE_SIZE, m_SpriteID, 0);
    if(m_State == Button::State::Hover && !Application::getIsPaused())
    {
        float         scale    = 35.0f;
        Vec2f         mousePos = Application::getCamera()->convertWindowToLevel(Event::getMousePos());
        std::string   name     = "Chest";
        uint8_t       layer    = 6;
        Render::hoverText(name, mousePos.x, mousePos.y, scale, {1.0f, 1.0f, 1.0f, 1.0f}, {0.3f, 0.3f, 0.3f, 0.7f},
     layer);
    }
}

void Chest::update()
{
    Vec2f mousePos = Application::getCamera()->convertWindowToLevel(Event::getMousePos());

    if(doesPointIntersectWithBox(mousePos, {x, y}, getCollisionBox()))
        m_State = Button::State::Hover;
    else
        m_State = Button::State::None;
}

bool Chest::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mouseClicked && !m_IsDud)
    {
        const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);

        Vec2f convPos = Application::getCamera()->convertWindowToLevel(ne.pos);

        Player *player = m_Level->getPlayer();
        if(doesPointIntersectWithBox(Application::getCamera()->convertWindowToLevel(ne.pos), {x, y},
    getCollisionBox()) && distBetweenVec2f({player->getX(), player->getY() - player->getWidth() / 2}, {x, y}) <
    1.5f * TILE_SIZE)
        {
            Event::ChangeGUIActiveLayer e1(InGameGUILayer::chestInventory);
            Application::callEvent(e1, true);

            Event::ChestOpenedEvent e2(&m_Inventory, nullptr, GUIInventoryIDCode::inventory);
            Application::callEvent(e2, true);
            return true;
        }
    }

    return Tile::eventCallback(e);
}
```

```
#ifdef DEBUG
void Chest::imGuiRender()
{
}
#endif
```

../src/maze/tiles/Chest.cpp

maze/tiles/SwitchTile.h

```
#pragma once

#include "Tile.h"
#include "Utils.h"

class SwitchTile : public Tile
{
  protected:
    Sprite::ID m_AltSpriteID;
    double     m_AltRotation;
    bool       showAlt;

  public:
    SwitchTile();
    SwitchTile(float x, float y, double rotation, Sprite::ID defaultSpriteID, double altRotation, Sprite::ID
    altSpriteID, bool isSolid, Level *level);
    virtual ~SwitchTile() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif

    virtual bool isSolid() override
    {
        return showAlt || m_IsSolid;
    }
};
```

../src/maze/tiles/SwitchTile.h

maze/tiles/SwitchTile.cpp

```
#include "SwitchTile.h"

SwitchTile::SwitchTile()
    : Tile(), m_AltSpriteID(Sprite::ID::errorID), showAlt(false)
{
}

SwitchTile::SwitchTile(float x, float y, double rotation, Sprite::ID defaultSpriteID, double altRotation, Sprite::
    ID altSpriteID, bool isSolid, Level *level)
    : Tile(x, y, rotation, defaultSpriteID, isSolid, level), m_AltSpriteID(altSpriteID), m_AltRotation(altRotation
    ), showAlt(false)
{
}

SwitchTile::~SwitchTile()
{
}

void SwitchTile::render()
{
    uint8_t layer = 0;
    if(showAlt)
        Render::sprite(x, y, m_AltRotation, TILE_SIZE, m_AltSpriteID, layer);
    else
        Render::sprite(x, y, rotation, TILE_SIZE, m_SpriteID, layer);
}

void SwitchTile::update()
{
}

bool SwitchTile::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::showAltTileEvent)
    {
        // TODO: Add particles
        const Event::ShowAltTileEvent &ne = static_cast<const Event::ShowAltTileEvent &>(e);

        showAlt = ne.showAlt;
    }
    return Tile::eventCallback(e);
}

#ifdef DEBUG
void SwitchTile::imGuiRender()
{
}
#endif
```

../src/maze/tiles/SwitchTile.cpp

maze/tiles/Trap.h

```
#pragma once

#include "Tile.h"

class Trap : public Tile
{
  protected:
    uint16_t m_AttackCooldown;
    uint16_t m_AttackCooldownMax;
    uint16_t m_AttackTrapTimer;
    Sprite::ID m_AttackSpriteID;
    uint32_t m_Damage;

    CollisionBox getCollisionBox();

  public:
    Trap();
    Trap(float x, float y, double rotation, Level *level);
    virtual ~Trap() override;

    virtual void render() override;
    virtual void update() override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif
};
```

../src/maze/tiles/Trap.h

maze/tiles/Trap.cpp

```
#include "Trap.h"

#include "Level.h"
#include "Utils.h"

Trap::Trap()
    : Tile(), m_AttackCooldown(0), m_AttackCooldownMax(30), m_AttackTrapTimer(0), m_AttackSpriteID(Sprite::ID::
    tileBasicTrapExposed), m_Damage(20)
{
}

Trap::Trap(float x, float y, double rotation, Level *level)
    : Tile(x, y, rotation, Sprite::ID::tileBasicTrapHidden, false, level), m_AttackCooldown(0),
    m_AttackCooldownMax(30), m_AttackTrapTimer(0), m_AttackSpriteID(Sprite::ID::tileBasicTrapExposed), m_Damage
    (20)
{
}

Trap::~Trap()
{
}

CollisionBox Trap::getCollisionBox()
{
    return {{-TILE_SIZE / 2, -TILE_SIZE / 2}, {TILE_SIZE / 2, TILE_SIZE / 2}};
}

void Trap::render()
{
    uint8_t layer = 0;
    if(m_AttackTrapTimer == 0)
        Render::sprite(x, y, rotation, TILE_SIZE, m_SpriteID, layer);
    else
        Render::sprite(x, y, rotation, TILE_SIZE, m_AttackSpriteID, layer);
```

```cpp
}

void Trap::update()
{
    if(m_AttackTrapTimer > 0)
        m_AttackTrapTimer--;
    if(m_AttackCooldown > 0)
        m_AttackCooldown--;
    else
    {
        // TODO: Make this work with all mobs
        Player *      player    = m_Level->getPlayer();
        CollisionBox &playerBox = player->getCollisionBox();
        if(doesPointIntersectWithBox({player->getX(), player->getY() - TILE_SIZE / 2}, {x, y}, getCollisionBox()))
        {
            m_AttackTrapTimer = 20;
            m_AttackCooldown  = m_AttackCooldownMax;
            player->dealDamage(m_Damage);
        }
    }
}

#ifdef DEBUG
void Trap::imGuiRender()
{
}
#endif
```

../src/maze/tiles/Trap.cpp

maze/tiles/Trapdoor.h

```cpp
#pragma once

#include "Button.h"
#include "Tile.h"
#include "Utils.h"

class Trapdoor : public Tile
{
  protected:
    Button::State m_State;

    CollisionBox getCollisionBox();

  public:
    Trapdoor();
    Trapdoor(float x, float y, double rotation, Level *level);
    virtual ~Trapdoor() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif
};
```

../src/maze/tiles/Trapdoor.h

maze/tiles/Trapdoor.cpp

```cpp
#include "Trapdoor.h"

#include "Application.h"
#include "FireStaff.h"
#include "Level.h"
#include "Player.h"
#include "Sprite.h"
#include "Utils.h"
#include "Player.h"

Trapdoor::Trapdoor()
    : Tile(), m_State(Button::State::None)
{
}

Trapdoor::Trapdoor(float x, float y, double rotation, Level *level)
    : Tile(x, y, rotation, Sprite::ID::tileBasicTrapdoor, false, level), m_State(Button::State::None)
{
}
```

```cpp
Trapdoor::~Trapdoor()
{
}

CollisionBox Trapdoor::getCollisionBox()
{
    return {{-TILE_SIZE / 2, -TILE_SIZE / 2}, {TILE_SIZE / 2, TILE_SIZE / 2}};
}

void Trapdoor::render()
{
    uint8_t layer = 0;
    Render::sprite(x, y, rotation, TILE_SIZE, m_SpriteID, layer);
    if(m_State == Button::State::Hover)
    {
        float         scale    = 35.0f;
        Vec2f         mousePos = Application::getCamera()->convertWindowToLevel(Event::getMousePos());
        std::string   name     = "Trapdoor";
        uint8_t       layer    = 6;
        Render::hoverText(name, mousePos.x, mousePos.y, scale, {1.0f, 1.0f, 1.0f, 1.0f}, {0.3f, 0.3f, 0.3f, 0.7f},
     layer);
    }
}

void Trapdoor::update()
{
    Vec2f mousePos = Application::getCamera()->convertWindowToLevel(Event::getMousePos());

    if(doesPointIntersectWithBox(mousePos, {x, y}, getCollisionBox()))
        m_State = Button::State::Hover;
    else
        m_State = Button::State::None;
}

bool Trapdoor::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mouseClicked && m_Level)
    {
        const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);

        Vec2f convPos = Application::getCamera()->convertWindowToLevel(ne.pos);

        Player *player = m_Level->getPlayer();
        if(doesPointIntersectWithBox(Application::getCamera()->convertWindowToLevel(ne.pos), {x, y},
    getCollisionBox()) && distBetweenVec2f({player->getX(), player->getY() - player->getWidth() / 2}, {x, y}) <
    1.5f * TILE_SIZE)
        {
            m_Level->endLevel();
            return true;
        }
    }

    return Tile::eventCallback(e);
}

#ifdef DEBUG
void Trapdoor::imGuiRender()
{
}
#endif
```

../src/maze/tiles/Trapdoor.cpp

3.7 Levels

levels/Level.h

```cpp
#pragma once

#include "Layer.h"

#include <functional>
#include <tuple>
#include <vector>

#include "KeyDefinitions.h"
#include "Utils.h"
#include "ItemContainer.h"
```

```cpp
#include "Entity.h"
#include "Item.h"
#include "Spawner.h"
#include "Player.h"
#include "Projectile.h"
#include "Room.h"
#include "Tile.h"

class Level : public Layer
{
  protected:
    int width, height;
    std::vector<Room *> m_Board;   // This stores Room * so that you can have the different subclasses of rooms
    also stored
    Vec2i               m_BoardOffset;

    Player                    m_Player;
    std::vector<Entity *>     m_Entities;
    std::vector<Projectile *> m_Projectiles;
    std::vector<Spawner *>    m_Spawners;

#ifdef DEBUG
    bool renderAll = false;
#endif

    bool collisionPointDetection(float nextX, float nextY);
    bool collisionTileDetection(int x, int y);
    bool lineCollisionDetection(float x, float y, float xs, float ys);
    std::tuple<bool, bool> lineCollisionCheck(float x, float y, float xs, float ys, bool returnFirst = false);

    int coordsToIndex(int x, int y);

    virtual void playerDeath() = 0;

  public:
    Level(float playerStartX, float playerStartY, int width, int height, Vec2i offsetStart);
    virtual ~Level();

    virtual void render();
    virtual void update();
    virtual bool eventCallback(const Event::Event &e);

#ifdef DEBUG
    virtual void imGuiRender() = 0;
#endif

    void addRoom(int x, int y, bool entrance[4], RoomType type);
    void removeRoom(int y, int x);

    void changeXOffset(int changeBy);
    void changeYOffset(int changeBy);

    float                       getX();
    float                       getY();
    Room *                      get(int y, int x);
    int                         getMidPoint();
    Room *                      getMidRoom();
    virtual Tile *              getTile(int x, int y);
    virtual Player *            getPlayer();
    virtual std::vector<Vec2f> *getPath(Vec2f startPos, Vec2f dest, CollisionBox box);

    std::tuple<Direction, Projectile *> getDirOfProjInRange(float x, float y, float range);

    float convertToRelativeX(float x);
    float convertToRelativeY(float y);
    Vec2f convertToRelativePos(Vec2f pos);
    bool  isOutOfBound(float x, float y);

    virtual void addEntity(Entity *e) { m_Entities.push_back(e); }
    virtual void addProjectile(Projectile *e) { m_Projectiles.push_back(e); }
    virtual void addSpawner(Spawner *e) { m_Spawners.push_back(e); }

    bool                collisionDetection(float nextX, float nextY, CollisionBox box);
    bool                directionalCollision(float x, float y, float xs, float ys, CollisionBox box);
    std::tuple<bool, bool> directionalCollisionCheck(float x, float y, float xs, float ys, CollisionBox box);
    virtual Entity *entityCollisionDetection(float nextX, float nextY, CollisionBox box);

    virtual void endLevel() = 0;
};
```

../src/levels/Level.h

levels/Level.cpp

```
#include "Level.h"

#include <algorithm>
#include <array>
#include <memory>

#include "AStarUtils.h"
#include "Application.h"
#include "Entity.h"
#include "Projectile.h"
#include "Room.h"
#include "Tile.h"

Level::Level(float playerStartX, float playerStartY, int width, int height, Vec2i offsetStart)
    : Layer(),
      width(width * ROOM_SIZE),
      height(height * ROOM_SIZE),
      m_BoardOffset(offsetStart),
      m_Player(playerStartX, playerStartY, this)
{
    m_Board.resize(width * height, nullptr);   // It is resized, because all positions are used straight away and
    fills any data slots will nullptr
    Application::getCamera()->setAnchor(&m_Player);
}

Level::~Level()
{
    for(int i = 0; i < m_Board.size(); i++)
    {
        if(m_Board[i])
            delete m_Board[i];
    }

    for(Entity *entity : m_Entities)
        delete entity;

    for(Projectile *projectile : m_Projectiles)
        delete projectile;

    for(Spawner *s : m_Spawners)
        delete s;
}

void Level::render()
{
    int midpoint = getMidPoint();
    Render::orderBuffersByYAxis();
#ifdef DEBUG
    if(renderAll)
    {   // This is to allow the option to render all - however only when debugging - because of the limit with
    vertices, they must be rendered in blocks
        int c = 0;
        for(int y = 0; y < MAZE_SIZE; y++)
        {
            for(int x = 0; x < MAZE_SIZE; x++)
            {
                Room *room = get(y, x);
                if(room)
                {
                    room->render();
                    c++;
                }
            }
        }
    }
    else
    {
        Room *mid = getMidRoom();
        mid->render();
        if(mid->isOpen(Direction::north) && get(midpoint + 1, midpoint))
            get(midpoint + 1, midpoint)->render();
        if(mid->isOpen(Direction::south) && get(midpoint - 1, midpoint))
            get(midpoint - 1, midpoint)->render();
        if(mid->isOpen(Direction::east) && get(midpoint, midpoint + 1))
            get(midpoint, midpoint + 1)->render();
        if(mid->isOpen(Direction::west) && get(midpoint, midpoint - 1))
            get(midpoint, midpoint - 1)->render();
    }
#else
    Room *mid = getMidRoom();
    mid->render();
    if(mid->isOpen(Direction::north))
        get(midpoint + 1, midpoint)->render();
    if(mid->isOpen(Direction::south))
        get(midpoint - 1, midpoint)->render();
    if(mid->isOpen(Direction::east))
        get(midpoint, midpoint + 1)->render();
    if(mid->isOpen(Direction::west))
        get(midpoint, midpoint - 1)->render();

#endif
    // Render::render(m_ShaderEffectsIDs);


    for(Entity *entity : m_Entities)
        entity->render();

    for(Projectile *projectile : m_Projectiles)
        projectile->render();

    for(Spawner *s : m_Spawners)
        s->render();

    m_Player.render();
}

void Level::update()
{
    m_Player.update();

    for(auto it = m_Entities.begin(); it != m_Entities.end();)
    {
        (*it)->update();
        if((*it)->deleteMe())
        {
            delete *it;
            it = m_Entities.erase(it);
        }
        else if(isOutOfBound((*it)->getX(), (*it)->getY()))
        {
            Mob *mob = dynamic_cast<Mob *>(*it);
            if(mob)
            {
                Application::callEventLater(new Event::MobDied(mob));
                ++it;
            }
            else
            {
                delete *it;
                it = m_Entities.erase(it);
            }
        }
        else
            ++it;
    }

    for(auto it = m_Projectiles.begin(); it != m_Projectiles.end();)
    {
        (*it)->update();
        if((*it)->deleteMe() || isOutOfBound((*it)->getX(), (*it)->getY()))
        {
            delete *it;
            it = m_Projectiles.erase(it);
        }

        else
            ++it;
    }

    for(auto it = m_Spawners.begin(); it != m_Spawners.end();)
    {
        (*it)->update();
        if((*it)->deleteMe() || isOutOfBound((*it)->getX(), (*it)->getY()))
        {
            delete *it;
            it = m_Spawners.erase(it);
        }
        else
            ++it;
    }

    int   midpoint = getMidPoint();
    Room *mid      = getMidRoom();
    mid->update();
    if(mid->isOpen(Direction::north))
        get(midpoint + 1, midpoint)->update();
    if(mid->isOpen(Direction::south))
        get(midpoint - 1, midpoint)->update();
    if(mid->isOpen(Direction::east))
        get(midpoint, midpoint + 1)->update();
    if(mid->isOpen(Direction::west))
        get(midpoint, midpoint - 1)->update();
}

bool Level::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        for(Room *room : m_Board)
        {
            if(room)
                room->eventCallback(e);
        }
    }
    else if(getMidRoom()->eventCallback(e))
        return true;

    if(m_Player.eventCallback(e))
        return true;

    for(Projectile *p : m_Projectiles)
    {
        if(p->eventCallback(e))
            return true;
    }

    for(Entity *entity : m_Entities)
    {
        if(entity->eventCallback(e))
            return true;
    }

    for(Spawner *s : m_Spawners)
    {
        if(s->eventCallback(e))
            return true;
    }

    if(e.getType() == Event::EventType::mobDied)
    {
        const Event::MobDied &ne = static_cast<const Event::MobDied &>(e);

        if(ne.mob == &m_Player)
            playerDeath();
        else
        {
            auto index = std::find(m_Entities.begin(), m_Entities.end(), ne.mob);
            if(index != m_Entities.end())
            {
                delete ne.mob;
                m_Entities.erase(index);
            }
        }
    }
    else if(e.getType() == Event::EventType::showAltTileEvent)
    {
        const Event::ShowAltTileEvent &ne = static_cast<const Event::ShowAltTileEvent &>(e);
        if(ne.showAlt)
        {
            auto dirToVec = [](Direction dir) -> Vec2f {
                Vec2f vec;
                switch(dir)
                {
                case Direction::north:
                    vec = {(ROOM_SIZE / 2) * TILE_SIZE, (ROOM_SIZE - 2.5f) * TILE_SIZE};
                    break;
                case Direction::south:
                    vec = {(ROOM_SIZE / 2) * TILE_SIZE, 1.5f * TILE_SIZE};
                    break;
                case Direction::east:
                    vec = {(ROOM_SIZE - 2.5f) * TILE_SIZE, (ROOM_SIZE / 2) * TILE_SIZE};
                    break;
                case Direction::west:
                    vec = {1.5f * TILE_SIZE, (ROOM_SIZE / 2) * TILE_SIZE};
                    break;
                default:
                    vec = {0.0f, 0.0f};
                    break;
                }
                return vec;
            };

            auto getChangeBy = [this, dirToVec](Vec2f startPos) -> Vec2f {
                Vec2f relPos = convertToRelativePos(startPos) - getMidPoint() * ROOM_PIXEL_SIZE;

                Direction shortestDir  = Direction::north;
                float     shortestDist = distBetweenVec2f(relPos, dirToVec(shortestDir));
                for(int dir = Direction::south; dir <= Direction::west; dir++)
                {
                    float dist = distBetweenVec2f(relPos, dirToVec(static_cast<Direction>(dir)));
                    if(dist < shortestDist)
                    {
                        shortestDir  = static_cast<Direction>(dir);
                        shortestDist = dist;
                    }
                }
                Vec2f pos = dirToVec(shortestDir);
                return pos - relPos;
            };

            Vec2f changeBy = getChangeBy({m_Player.getX(), m_Player.getY()});
            m_Player.changeX(changeBy.x);
            m_Player.changeY(changeBy.y);

            for(Entity *e : m_Entities)
            {
                Vec2f changeBy = getChangeBy({e->getX(), e->getY()});
                e->changeX(changeBy.x);
                e->changeY(changeBy.y);
            }
        }
    }

    return false;
}

void Level::addRoom(int x, int y, bool entrances[4], RoomType type)
{
    if(x < 0 || x >= MAZE_SIZE || y < 0 || y >= MAZE_SIZE)
    {
        Log::error("Tried to create room out of bounds", LOGINFO);
        return;
    }

    m_Board[coordsToIndex(x, y)] = new Room(getX() + x * ROOM_PIXEL_SIZE, getY() + y * ROOM_PIXEL_SIZE, entrances,
     type, this);
}

void Level::removeRoom(int y, int x)
{
    if(x < 0 || x >= MAZE_SIZE || y < 0 || y >= MAZE_SIZE)
    {
        Log::error("Tried to delete room out of bounds", LOGINFO);
        return;
    }
    delete m_Board[coordsToIndex(x, y)];
    m_Board[coordsToIndex(x, y)] = nullptr;
}

Room * Level::get(int y, int x)
{
    int index = coordsToIndex(x, y);
    if(index == -1)
        return nullptr;
    return m_Board[coordsToIndex(x, y)];
```

```
}

int Level::getMidPoint()
{
    return MAZE_SIZE / 2;
}

Room *Level::getMidRoom()
{
    return get(getMidPoint(), getMidPoint());
}

Tile *Level::getTile(int x, int y)
{
    int roomX = x / ROOM_SIZE - m_BoardOffset.x;
    int tileX = x % ROOM_SIZE;
    int roomY = y / ROOM_SIZE - m_BoardOffset.y;
    int tileY = y % ROOM_SIZE;

    Room *room = get(roomY, roomX);
    if(!room)
    {
        // Log::warning("Trying to access room that doesn't exist!");
        return nullptr;
    }

    return room->getTile(tileX, tileY);
}

Player *Level::getPlayer()
{
    return &m_Player;
}

int Level::coordsToIndex(int x, int y)
{
    if(x < 0 || x >= MAZE_SIZE || y < 0 || y >= MAZE_SIZE)
        return -1;
    int xCoord = x + m_BoardOffset.x;
    int yCoord = y + m_BoardOffset.y;
    if(xCoord >= MAZE_SIZE)
        xCoord -= MAZE_SIZE;
    if(yCoord >= MAZE_SIZE)
        yCoord -= MAZE_SIZE;
    return yCoord * MAZE_SIZE + xCoord;
}



void Level::changeXOffset(int changeBy)
{
    m_BoardOffset.x += changeBy;

    if(m_BoardOffset.x == MAZE_SIZE)
    {
        m_BoardOffset.x = 0;

        Event::MazeMovedEvent e((float) -ROOM_PIXEL_SIZE * MAZE_SIZE, 0.0f);
        Application::callEvent(e, true);
    }
    else if(m_BoardOffset.x == -1)
    {
        m_BoardOffset.x = MAZE_SIZE - 1;
        Event::MazeMovedEvent e((float) ROOM_PIXEL_SIZE * MAZE_SIZE, 0.0f);
        Application::callEvent(e, true);
    }
}

void Level::changeYOffset(int changeBy)
{
    m_BoardOffset.y += changeBy;

    if(m_BoardOffset.y == MAZE_SIZE)
    {
        m_BoardOffset.y = 0;
        Event::MazeMovedEvent e(0.0f, (float) -ROOM_PIXEL_SIZE * MAZE_SIZE);
        Application::callEvent(e, true);
    }
    else if(m_BoardOffset.y == -1)
    {
        m_BoardOffset.y = MAZE_SIZE - 1;
        Event::MazeMovedEvent e(0.0f, (float) ROOM_PIXEL_SIZE * MAZE_SIZE);
        Application::callEvent(e, true);
    }
}

float Level::getX()
{
    return m_BoardOffset.x * ROOM_PIXEL_SIZE;
}

float Level::getY()
{
    return m_BoardOffset.y * ROOM_PIXEL_SIZE;

}

std::vector<Vec2f> *Level::getPath(Vec2f startPos, Vec2f destPos, CollisionBox box)
{
    Vec2f relativeStart = convertToRelativePos(startPos);
    Vec2f relativeDest = convertToRelativePos(destPos);

    std::vector<Vec2f> *path = new std::vector<Vec2f>();

    // A* on rooms
    Vec2i startRoom = {(int) relativeStart.x / ROOM_PIXEL_SIZE, (int) relativeStart.y / ROOM_PIXEL_SIZE};
    Vec2i destRoom  = {(int) relativeDest.x / ROOM_PIXEL_SIZE, (int) relativeDest.y / ROOM_PIXEL_SIZE};

    if(startRoom != destRoom)
    {
        if(!get(startRoom.y, startRoom.x) || !get(destRoom.y, destRoom.x))
        {
            Log::warning("Room that destination or start does not exist!");
            std::vector<Vec2f> *path = new std::vector<Vec2f>();
            path->push_back(destPos);
            return path;
        }

        std::array<Vec2i, 4> offsets;
        offsets[0] = {0, 1};
        offsets[1] = {0, -1};
        offsets[2] = {1, 0};
        offsets[3] = {-1, 0};

        std::function<bool(int, int, int, int, CollisionBox)> collisionDetection = [this](int x, int y, int xs,
    int ys, CollisionBox box) -> bool {
            if(!get(y + ys, x + xs) /* || !get(y, x)*/)
                return true;
            else if(ys == 1)
                return !get(y, x)->isOpen(Direction::north);
            else if(ys == -1)
                return !get(y, x)->isOpen(Direction::south);
            else if(xs == 1)
                return !get(y, x)->isOpen(Direction::east);
            else if(xs == -1)
                return !get(y, x)->isOpen(Direction::west);
            else
                return false;
        };
        std::function<Vec2f(Vec2i)> convert = [](Vec2i vec) -> Vec2f {
            return {(float) vec.x * ROOM_PIXEL_SIZE, (float) vec.y * ROOM_PIXEL_SIZE};
        };
        std::vector<Vec2f> *myPath = aStarAlgorithm<MAZE_SIZE, MAZE_SIZE, 4>(startRoom, destRoom, box, offsets,
    collisionDetection, convert, MAZE_SIZE * MAZE_SIZE);

        float mid = ((float) TILE_SIZE * ROOM_SIZE) / 2;
        relativeDest      = {(float) (*myPath)[myPath->size() - 1].x + mid, (float) (*myPath)[myPath->size() - 1].
    y + mid};

        delete myPath;
    }

    // A* on tiles

    // These are the coordinates to the nearest node (on the whole board)
    Vec2i startNode = {(int) round(relativeStart.x / X_STEP), (int) round(relativeStart.y / Y_STEP)};
    Vec2i destNode  = {(int) round(relativeDest.x / X_STEP), (int) round(relativeDest.y / Y_STEP)};

    if(collisionDetection(destNode.x * X_STEP + getX(), destNode.y * Y_STEP + getY(), box))
    {
        // TODO: Put this in a separate function
        bool foundAlternative = false;

        for(int x = -1; x < 2; x++)
        {
            for(int y = -1; y < 2; y++)
            {
                if(x == 0 && y == 0)
                    continue;
                if(!collisionDetection((destNode.x + x) * X_STEP + getX(), (destNode.y + y) * Y_STEP + getY(), box
    ))
                {
                    destNode.x += x;
                    destNode.y += y;
                    foundAlternative = true;
                    break;
                }
            }
        }

        if(!foundAlternative)
        {
            Log::warning("Cannot reach destNode!");
            path->push_back(destPos);
            return path;
        }
    }
    if(startNode == destNode)
    {
        path->push_back(destPos);
        return path;
    }

    // These positions are relative to the grid used in the A* algorithm
    Vec2i gridStartPos = {X_MAX / X_STEP, Y_MAX / Y_STEP};
    Vec2i gridDestPos  = {destNode.x - startNode.x + gridStartPos.x, destNode.y - startNode.y + gridStartPos.y};

    std::array<Vec2i, 8> offsets;
    offsets[0] = {-1, 1};
    offsets[1] = {0, 1};
    offsets[2] = {1, 1};
    offsets[3] = {1, 0};
    offsets[4] = {1, -1};
    offsets[5] = {0, -1};
    offsets[6] = {-1, -1};
    offsets[7] = {-1, 0};

    std::function<Vec2f(Vec2i)> convert = [gridStartPos, startNode, this](Vec2i vec) -> Vec2f {
        return {(float) (vec.x - gridStartPos.x + startNode.x) * X_STEP + getX(),
                (float) (vec.y - gridStartPos.y + startNode.y) * Y_STEP + getY()};
    };

    std::function<bool(int, int, int, int, CollisionBox)> collisionDetection = [this, &convert](int x, int y, int
    xs, int ys, CollisionBox box) -> bool {
        Vec2f pos = convert({x, y});
        return directionalCollision(pos.x, pos.y, xs * X_STEP, ys * Y_STEP, box);
    };

    Vec2f checkDest = {(float) destNode.x * X_STEP + getX(),
                       (float) destNode.y * Y_STEP + getY()};

    if(convert(gridDestPos) != checkDest)
        Log::critical("Convertion does not result in the correct value!", LOGINFO);

    // Log::info("Level A*");

    return aStarAlgorithm<2 * X_MAX / X_STEP, 2 * Y_MAX / Y_STEP, 8>(gridStartPos, gridDestPos, box, offsets,
    collisionDetection, convert, X_MAX / X_STEP);
}

std::tuple<Direction, Projectile *> Level::getDirOfProjInRange(float x, float y, float range)
{
    Direction   closestDir  = Direction::north;
    Projectile *closestProj = nullptr;
    float       closestDist = range;
    for(Projectile *proj : m_Projectiles)
    {
        float dist = distBetweenVec2f({x, y}, {proj->getX(), proj->getY()});
        if(dist < closestDist)
        {
```

```cpp
            Direction dir   = proj->getDirection();
            float     distX = proj->getX() - x;
            float     distY = proj->getY() - y;

            // This checks if the projectile is going towards the center (ignoring collisions)
            if(std::fabs(distX) > std::fabs(distY))
            {
                if((distX < 0 && dir == Direction::east) || (distX > 0 && dir == Direction::west))
                {
                    closestDir  = dir;
                    closestProj = proj;
                }
            }
            else if(std::fabs(distX) > std::fabs(distY))
            {
                if((distY < 0 && dir == Direction::north) || (distY > 0 && dir == Direction::south))
                {
                    closestDir  = dir;
                    closestProj = proj;
                }
            }
            else   // TODO: Check if this is going the right way
            {
                closestDir  = dir;
                closestProj = proj;
            }
        }
    }
    return {closestDir, closestProj};
}

float Level::convertToRelativeX(float x)
{
    return x - getX();
}

float Level::convertToRelativeY(float y)
{
    return y - getY();

}

Vec2f Level::convertToRelativePos(Vec2f pos)
{
    return {convertToRelativeX(pos.x), convertToRelativeY(pos.y)};
}

bool Level::isOutOfBound(float x, float y)
{
    Vec2f pos = convertToRelativePos({x, y});
    return pos.x < 0 || pos.x > width * TILE_SIZE || pos.y < 0 || pos.y > height * TILE_SIZE;
}

bool Level::collisionPointDetection(float nextX, float nextY)
{
    int   tileX = (int) nextX / TILE_SIZE;
    int   tileY = (int) nextY / TILE_SIZE;
    Tile *tile  = getTile(tileX, tileY);
    if(!tile)
        return true;

    return tile->isSolid();
}

bool Level::collisionTileDetection(int x, int y)
{
    Tile *tile = getTile(x, y);
    if(!tile)
        return true;

    return tile->isSolid();
}

bool Level::collisionDetection(float nextX, float nextY, CollisionBox box)
{
    if(isOutOfBound(nextX + box.lowerBound.x, nextY + box.lowerBound.y) || isOutOfBound(nextX + box.upperBound.x,
    nextY + box.upperBound.y))
        return true;

    bool lowerLeft  = collisionPointDetection(nextX + box.lowerBound.x, nextY + box.lowerBound.y);
    bool lowerRight = collisionPointDetection(nextX + box.upperBound.x, nextY + box.lowerBound.y);
    bool upperLeft  = collisionPointDetection(nextX + box.lowerBound.x, nextY + box.upperBound.y);
    bool upperRight = collisionPointDetection(nextX + box.upperBound.x, nextY + box.upperBound.y);

    return lowerLeft || lowerRight || upperLeft || upperRight;
}

bool Level::directionalCollision(float x, float y, float xs, float ys, CollisionBox box)
{
    auto [colX, colY] = directionalCollisionCheck(x, y, xs, ys, box);
    return colX || colY;
}

std::tuple<bool, bool> Level::directionalCollisionCheck(float x, float y, float xs, float ys, CollisionBox box)
{
    if(xs == 0 && ys == 0)
        return {true, true};
    else
    {
        if(isOutOfBound(x + box.lowerBound.x, y + box.lowerBound.y) || isOutOfBound(x + box.upperBound.x, y + box.
    upperBound.y))
            return {true, true};

        auto checks = [this](float x, float y, float xs, float ys, Vec2f(&posOffsets)[3]) -> std::tuple<bool, bool
    > {
            bool colX = false;
            bool colY = false;
            for(int i = 0; i < 3; i++)
            {
                if(posOffsets[i].x == 0.0f && posOffsets[i].y == 0.0f)
                    continue;

                auto [tempX, tempY] = lineCollisionCheck(x + posOffsets[i].x, y + posOffsets[i].y, xs, ys, true);

                if(tempX)
                    colX = true;
                if(tempY)
                    colY = true;
                if(colX && colY)
                    break;
            }
            return {colX, colY};
        };

        if((xs > 0 && ys > 0) || (xs < 0 && ys < 0))   // Travelling along a line close to this: / path
        {
            Vec2f offsets[3] = {
                {box.upperBound.x, box.lowerBound.y},   // Lower Right
                {box.lowerBound.x, box.upperBound.y},   // Upper Left
                {0.0f, 0.0f}                            // Middle
            };

            // Defining the middle
            if(xs > 0)   // Upper right corner
                offsets[2] = {box.upperBound.x, box.upperBound.y};
            else   // Lower left corner
                offsets[2] = {box.lowerBound.x, box.lowerBound.y};

            return checks(x, y, xs, ys, offsets);
        }
        else if((xs > 0 && ys < 0) || (xs < 0 && ys > 0))   // Travelling along a line close to this: \ path
        {
            Vec2f offsets[3] = {
                {box.lowerBound.x, box.lowerBound.y},   // Lower Left
                {box.upperBound.x, box.upperBound.y},   // Upper Right
                {0.0f, 0.0f}                            // Middle
            };

            // Defining the middle
            if(xs > 0)   // Lower right corner
                offsets[2] = {box.upperBound.x, box.lowerBound.y};
            else   // Upper left corner
                offsets[2] = {box.lowerBound.x, box.upperBound.y};

            return checks(x, y, xs, ys, offsets);
        }
        else if(xs != 0)   // Travelling along a line parallel to ---
        {
            Vec2f offsets[3] = {
                {xs > 0 ? box.upperBound.x : box.lowerBound.x, box.upperBound.y},   // Upper
                {xs > 0 ? box.upperBound.x : box.lowerBound.x, box.lowerBound.y},   // Lower
                {0.0f, 0.0f}                                                        // No checks for middle
            };
            return checks(x, y, xs, ys, offsets);
        }
        else   // Travelling along a line parallel to |
        {
            Vec2f offsets[3] = {
                {box.upperBound.x, ys > 0 ? box.upperBound.y : box.lowerBound.y},   // Upper
                {box.lowerBound.x, ys > 0 ? box.upperBound.y : box.lowerBound.y},   // Lower
                {0.0f, 0.0f}                                                        // No checks for middle
            };
            return checks(x, y, xs, ys, offsets);
        }
}
}

Entity *Level::entityCollisionDetection(float nextX, float nextY, CollisionBox box)
{
    if(m_Player.hasCollidedWith(nextX, nextY, box))
        return &m_Player;

    for(Entity *e : m_Entities)
    {
        if(e->hasCollidedWith(nextX, nextY, box))
            return e;
    }

    return getMidRoom()->entityCollisionDetection(nextX, nextY, box);
}

bool Level::lineCollisionDetection(float x, float y, float xs, float ys)
{
    auto [colX, colY] = lineCollisionCheck(x, y, xs, ys);

    return colX || colY;
}

std::tuple<bool, bool> Level::lineCollisionCheck(float x, float y, float xs, float ys, bool returnFirst)
{
    bool noStep = collisionPointDetection(x, y);
    if(noStep || (xs == 0 && ys == 0))
        return {noStep, noStep};

    // this is a simple function to return what sign a given float has
    // This is used so as a simple way to determine the direction of travel
    auto getSign = [](float check) -> int {
        return check > 0 ? 1 : -1;
    };

    auto getPer = [](float relPos, float change) {
        // This calculates the percentage of the way through the tile boundary is hit, and thus the first boundary
     to be hit
        if(change > 0)
        {
            if(relPos + change > TILE_SIZE)
                return (TILE_SIZE - relPos) / change;
        }
        else
        {   // This means change is negative so it has to check it doesn't go passed 0
            if(relPos + change < 0.0f)
                return relPos / change;
        }
        // Set to 1.1 so that if it doesn't hit another boundary it does not interfere with the comparason
        return 1.1f;
    };

    bool crossedX = false;
    bool crossedY = false;

    // This makes the coordinates relative to the current the start of the line is on, making it easier for
    calculations later
    int   tileX     = (int) x / TILE_SIZE;
    int   tileY     = (int) y / TILE_SIZE;
    float relativeX = x - tileX * TILE_SIZE;
    float relativeY = y - tileY * TILE_SIZE;

    if(relativeX + xs < TILE_SIZE && relativeX + xs > 0 && relativeY + ys < TILE_SIZE && relativeY + ys > 0)
        return {false, false};
```

```cpp
    int xOffset = 0;
    int yOffset = 0;
    // This continues running until the xs or ys does not travel over a tile boundary
    while(relativeX + xs > TILE_SIZE || relativeY + ys > TILE_SIZE || relativeX + xs < 0 || relativeY + ys < 0)
    {
        float xPer = getPer(relativeX, xs);
        float yPer = getPer(relativeY, ys);

        if(xPer < yPer)   // This just goes through and checks with tile to check
        {
            xOffset += getSign(xs);
            if(collisionTileDetection(tileX + xOffset, tileY + yOffset))
            {
                if(returnFirst)
                    return {true, false};
                crossedX = true;
            }
            relativeX -= getSign(xs) * TILE_SIZE;
        }
        else if(yPer < xPer)
        {
            yOffset += getSign(ys);
            if(collisionTileDetection(tileX + xOffset, tileY + yOffset))
            {
                if(returnFirst)
                    return {false, true};
                crossedY = true;
            }
            relativeY -= getSign(ys) * TILE_SIZE;
        }
        else   // This means that it passes through the corner
        {
            if(collisionTileDetection(tileX + xOffset + getSign(xs), tileY + xOffset + getSign(ys)))
                return {true, true};

            if(collisionTileDetection(tileX + xOffset + getSign(xs), tileY + yOffset))
            {
                if(returnFirst)
                    return {true, false};
                crossedX = true;
            }
            if(collisionTileDetection(tileX + xOffset, tileY + yOffset + getSign(ys)))
            {
                if(returnFirst)
                    return {false, true};
                crossedY = true;
            }

            relativeX -= getSign(xs) * TILE_SIZE;
            relativeY -= getSign(ys) * TILE_SIZE;

            xOffset += getSign(xs);
            yOffset += getSign(ys);
        }
        if(crossedX && crossedY)
            break;
    }

    return {crossedX, crossedY};
}
```

../src/levels/Level.cpp

levels/Maze.h

```cpp
#pragma once

#include "Camera.h"
#include "GUILayer.h"
#include "Level.h"
#include "Renderer.h"
#include "Room.h"

#include "AnimatedSprite.h"
#include "MazeHeaders.h"

#include "Player.h"

#include "KeyDefinitions.h"

class Maze : public Level
```

```
{
  protected:

    // This is for multithreading - and are the variables that allow threads to communicate
    bool finishedGenerating;

    std::vector<Vec2i> currentPaths;   // This stores the current possible paths

    // This stores the entrance state, allowing the possibility of forcing an entrance later on after generatings
    enum EntranceState
    {
        couldOpen = 0,
        isOpen    = 1,
        isClosed  = 2
    };

    bool pathsNorth[MAZE_SIZE];   // This stores the avaliable paths for each direction, allowing generation when
    the player moves
    bool pathsSouth[MAZE_SIZE];
    bool pathsEast[MAZE_SIZE];
    bool pathsWest[MAZE_SIZE];

    uint64_t moves;
    bool     isMoving[4];

    void playerMoved(Direction dir);

    void addRoom(int x, int y, bool north, bool south, bool east, bool west);
    void updatePaths();

    void multithreadGenerating(int layerMax, int startMax);
    void generatePaths(int layerMax, int startMax);

    EntranceState shouldBeOpen(Room *room, Direction nextEntrance, int prob, int *pathCount);
    void          forceEntrance(EntranceState *north, EntranceState *south, EntranceState *east, EntranceState *
    west);

    // TODO: Make this a event
    virtual void endLevel() override;
    virtual void playerDeath() override;
    void resetMaze();

  public:
    Maze();
    ~Maze();

    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif

    void generate();
    void moveNorth();
    void moveSouth();
    void moveEast();
    void moveWest();
};
```

../src/levels/Maze.h

levels/Maze.cpp

```
#include "Maze.h"

#include <ImGui.h>
#include <algorithm>
#include <limits>
#include <math.h>
#include <string>

#include "AStarUtils.h"
#include "Application.h"
#include "Log.h"
#include "RandomGen.h"
#include "VertexBufferLayout.h"

#include "GUIStack.h"

#include "NPC.h"
#include "Room.h"
```

```
#include "Tile.h"

#include "WorldItem.h"

#include "FireStaff.h"
#include "Potion.h"

#define LAYER_MAX_FOR_DIRECTIONS 4

#include <vector>

#define MAZE_MIDPOINT ((float) MAZE_SIZE / 2.0f) * ROOM_PIXEL_SIZE - TILE_SIZE / 2

Maze::Maze()
    : Level(MAZE_MIDPOINT, MAZE_MIDPOINT, MAZE_SIZE, MAZE_SIZE, {0, 0}),
      finishedGenerating(true),
      moves(0)
{
    for(int i = 0; i < 4; i++)
        isMoving[i] = false;

    // NOTE: Because of how it is rendering the coords (0,0) on the m_Board is the bottom left, not the top left!!

    currentPaths.reserve(2 * MAZE_SIZE);    // The data is reserved here because not all the data is needed, but it
     could be used and so for efficiency, it is reserved on init

    NPC *follower = new NPC(3100.0f, 3800.0f, this, NPC::Type::Follower, NPC::Race::Fire);
    m_Player.addFollower(follower);
    m_Entities.push_back(follower);

    Application::addOverlay(new GUIStack(this));

    Log::info("Maze initialised");
}

Maze::~Maze()
{
    // NOTE: This needs to be caused before the progam ends as it frees up the memory
    Log::info("Maze destroyed");
}

void Maze::update()
{
    if(!finishedGenerating)
        return;

    if(finishedGenerating)
    {
        bool noEntrances = true;
        for(int i = 0; i < MAZE_SIZE; i++)
        {
            if(pathsNorth[i] || pathsSouth[i] || pathsEast[i] || pathsWest[i])
            {
                noEntrances = false;
                break;
            }
        }

        if(noEntrances)
            resetMaze();
    }

    Vec2f playerPos = convertToRelativePos({m_Player.getX(), m_Player.getY()});
    if(playerPos.y > (getMidPoint() + 1) * TILE_SIZE * ROOM_SIZE)
        playerMoved(Direction::north);
    else if(playerPos.y < getMidPoint() * TILE_SIZE * ROOM_SIZE)
        playerMoved(Direction::south);

    if(playerPos.x > (getMidPoint() + 1) * TILE_SIZE * ROOM_SIZE)
        playerMoved(Direction::east);
    else if(playerPos.x < getMidPoint() * TILE_SIZE * ROOM_SIZE)
        playerMoved(Direction::west);

    Level::update();
}

#ifdef DEBUG
void Maze::imGuiRender()
{
    if(ImGui::Button("Reload Maze"))
    {
        // TODO: Change player coords
        generate();
    }
    ImGui::Checkbox("Render all", &renderAll);
    m_Player.imGuiRender();
}
#endif

bool Maze::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::keyInput)
    {
        const Event::KeyboardEvent &ne = static_cast<const Event::KeyboardEvent &>(e);
        if(ne.key == GLFW_KEY_E && ne.action == GLFW_PRESS)
        {
            Event::ChangeGUIActiveLayer e(InGameGUILayer::playerInventory);
            Application::callEvent(e, true);

            return true;
        }
        else if(ne.key == GLFW_KEY_ESCAPE && ne.action == GLFW_PRESS)
        {
            Event::ChangeGUIActiveLayer e(InGameGUILayer::exitMenu);
            Application::callEvent(e, true);

            return true;
        }
    }


    return Level::eventCallback(e);
}

void Maze::endLevel()
{
    Event::ChangeGUIActiveLayer e(InGameGUILayer::playerWin);
    Application::callEvent(e, true);
}

void Maze::playerDeath()
{
    Log::info("Player has died");
    m_Player.resetStats();
    m_Player.changeX(3800.0f - m_Player.getX());
    m_Player.changeY(3800.0f - m_Player.getY());
    resetMaze();
}

void Maze::resetMaze()
{
    Log::info("Resetting the maze!");
    for(Entity *e : m_Entities)
        delete e;
    m_Entities.clear();

    for(Projectile *p : m_Projectiles)
        delete p;
    m_Projectiles.clear();

    for(int i = 0; i < MAZE_SIZE * MAZE_SIZE; i++)
    {
        if(m_Board[i])
        {
            delete m_Board[i];
            m_Board[i] = nullptr;
        }
    }

    generate();
}

// SECTION: Rooms
void Maze::addRoom(int x, int y, bool north, bool south, bool east, bool west)
{
    bool entrances[4]           = {north, south, east, west};
    RoomType type                   = RoomType::Empty;

    float graphX  = std::abs(x - MAZE_SIZE / 2) + std::abs(y - MAZE_SIZE / 2) + moves;
    auto  sigmoid = [graphX](float lim, float decay) -> int {
        return round((2 * lim) / (1 + exp(-decay * graphX / 5.0f)) - lim);
    };

    // TODO: Make this based on the distance moved
    int num = Random::getNum(1, 100);
    if(num == sigmoid(1, 0.5f))
        type = RoomType::Exit;
    else if(num < sigmoid(7, 1))
        type = RoomType::Enemy;    // NOTE: This was enemy
    else if(num < sigmoid(12, 1))
        type = RoomType::NPC;
    else if(num < sigmoid(22, 1))
        type = RoomType::Trap;
    else if(num < sigmoid(32, 2))
        type = RoomType::Chest;

    Level::addRoom(x, y, entrances, type);
}

void Maze::playerMoved(Direction dir)
{
    if(isMoving[dir])
    {
        isMoving[dir] = false;

        if(moves >= std::numeric_limits<uint64_t>::max() - 2)
            moves = 0;
        else
            moves += 2;
    }
    else
        isMoving[dir] = true;

    switch(dir)
    {
    case Direction::north:
        Log::info("Player moved north");
        isMoving[Direction::south] = false;
        moveNorth();
        break;
    case Direction::south:
        Log::info("Player moved south");
        isMoving[Direction::north] = false;
        moveSouth();
        break;
    case Direction::east:
        Log::info("Player moved east");
        isMoving[Direction::west] = false;
        moveEast();
        break;
    case Direction::west:
        Log::info("Player moved west");
        isMoving[Direction::east] = false;
        moveWest();
        break;
    default:
        Log::warning("Unknown direction! When generating");
        break;
    }

    getMidRoom()->active();
}

// !SECTION

// SECTION: Generation
void Maze::generatePaths(int layerMax, int startMax)
{
    // Log::info("Generating paths");
    int layer = 0;

    // Resets the paths north
    for(int i = 0; i < MAZE_SIZE; i++)
    {
        pathsNorth[i] = false;
        pathsSouth[i] = false;
        pathsEast[i]  = false;
        pathsWest[i]  = false;
    }
```

```
/*
        This maze generation works using a tree style method, where each branch or path is generated one at a
time, and then
        any open entrances that is left is also then added to the currentPaths vector, ready to be generated
in the next layer.
        TODO: Add the ability to go back, if not many rooms have been created and look for spaces some can be
    */

std::vector<Vec2i> newPaths;   // This stores the newPaths for the next layer
newPaths.reserve(MAZE_SIZE);   // Reserves space because, not all spaces may be used
while(currentPaths.size() > 0)
{
    for(int i = 0; i < currentPaths.size(); i++)
    {   // This goes through all the current path options and generates the options for the rooms
        Vec2i pos = currentPaths[i];

        if(get(pos.y, pos.x))   // Checks that the pointer is a nullptr - so it doesn’t overwrite any rooms
            continue;

        int prob = startMax;   // This sets the probability of for the chance of generating an entrance
        if(pos.x > (3 * MAZE_SIZE / 4 + 1) || pos.x < MAZE_SIZE / 4 + 1)
        {
            prob++;
        }
        else if(pos.y > (3 * MAZE_SIZE / 4 + 1) || pos.y < MAZE_SIZE / 4 + 1)
        {
            prob++;
        }

        int pathCount = 0;   // Stores the number of paths from that room

        // 0: closed but could be open, 1: open, 2: closed and cannot be open
        EntranceState north = shouldBeOpen(get(pos.y + 1, pos.x), Direction::south, prob, &pathCount);
        EntranceState south = shouldBeOpen(get(pos.y - 1, pos.x), Direction::north, prob, &pathCount);
        EntranceState east  = shouldBeOpen(get(pos.y, pos.x + 1), Direction::west, prob, &pathCount);
        EntranceState west  = shouldBeOpen(get(pos.y, pos.x - 1), Direction::east, prob, &pathCount);

        // To increase the spread for the beginning part of generation, if only one entrance has been
generated -
        // then it will force another entrance if the layer is still below the layerMax
        if(pathCount <= 1 && layer < layerMax)
        {
            forceEntrance(&north, &south, &east, &west);

            int r = Random::getNum(0, 2);
            if(r != 2)
            {
                forceEntrance(&north, &south, &east, &west);
            }
        }
        else if(pathCount == 2 && layer < layerMax - MAZE_SIZE / 3)
        {
            int r = Random::getNum(0, 2);
            if(r != 2)
            {
                forceEntrance(&north, &south, &east, &west);
            }
        }

        // This is to check if any errors have occurred when generating the maze
        if(north == EntranceState::isOpen && pos.y < MAZE_SIZE - 1 && get(pos.y + 1, pos.x) && !get(pos.y + 1,
 pos.x)->isOpen(Direction::south))
            Log::error("Room generated incorrectly!", LOGINFO);
        if(south == EntranceState::isOpen && pos.y > 0 && get(pos.y - 1, pos.x) && !get(pos.y - 1, pos.x)->
isOpen(Direction::north))
            Log::error("Room generated incorrectly!", LOGINFO);
        if(east == EntranceState::isOpen && pos.x < MAZE_SIZE - 1 && get(pos.y, pos.x + 1) && !get(pos.y, pos.
x + 1)->isOpen(Direction::west))
            Log::error("Room generated incorrectly!", LOGINFO);
        if(west == EntranceState::isOpen && pos.x > 0 && get(pos.y, pos.x - 1) && !get(pos.y, pos.x - 1)->
isOpen(Direction::east))
            Log::error("Room generated incorrectly!", LOGINFO);

        // This adds any new paths made to the newPaths list
        if(north == EntranceState::isOpen && pos.y < MAZE_SIZE - 1 && !get(pos.y + 1, pos.x))
            newPaths.push_back({pos.x, pos.y + 1});
        if(south == EntranceState::isOpen && pos.y > 0 && !get(pos.y - 1, pos.x))
            newPaths.push_back({pos.x, pos.y - 1});
        if(east == EntranceState::isOpen && pos.x < MAZE_SIZE - 1 && !get(pos.y, pos.x + 1))
            newPaths.push_back({pos.x + 1, pos.y});
            if(west == EntranceState::isOpen && pos.x > 0 && !get(pos.y, pos.x - 1))
                newPaths.push_back({pos.x - 1, pos.y});

            addRoom(pos.x, pos.y,
                    north == EntranceState::isOpen,
                    south == EntranceState::isOpen,
                    east == EntranceState::isOpen,
                    west == EntranceState::isOpen);
        }
        currentPaths = newPaths;
        newPaths.clear();

        layer++;
    }

    updatePaths();

    finishedGenerating = true;
}

void Maze::multithreadGenerating(int layerMax, int startMax)
{
    if(!finishedGenerating)
        Log::critical("Stacked maze generating!!", LOGINFO);
    finishedGenerating = false;
    std::thread t1(&Maze::generatePaths, this, layerMax, startMax);   // This starts the multithreading
    t1.detach();
}

void Maze::generate()
{
    for(int i = 0; i < MAZE_SIZE * MAZE_SIZE; i++)
    {
        if(m_Board[i])
        {
            delete m_Board[i];   // NOTE: This frees up the memory, but does not make it a nullptr
            m_Board[i] = nullptr;
        }
    }
    m_BoardOffset.y = 0;
    m_BoardOffset.x = 0;
    int midpoint    = MAZE_SIZE / 2;
    // NOTE: MUST DELETE ALL ROOMS!

    bool entrances[4] = {true, true, true, true};
    Level::addRoom(midpoint, midpoint, entrances, RoomType::Empty);
    currentPaths.push_back({midpoint - 1, midpoint});
    currentPaths.push_back({midpoint, midpoint - 1});
    currentPaths.push_back({midpoint + 1, midpoint});
    currentPaths.push_back({midpoint, midpoint + 1});

    Item *     item      = new FireStaff();
    WorldItem *worldItem = new WorldItem(3800.0f, 3800.0f, TILE_SIZE / 2, this, item);
    getMidRoom()->addEntity(worldItem);
    Item *     item2      = new FireStaff();
    WorldItem *worldItem2 = new WorldItem(3900.0f, 3800.0f, TILE_SIZE / 2, this, item2);
    getMidRoom()->addEntity(worldItem2);

    Item *     potion     = new Potion(Potion::Type::HealthHuge);
    WorldItem *worldItem3 = new WorldItem(3800.0f, 3900.0f, TILE_SIZE / 2, this, potion);
    getMidRoom()->addEntity(worldItem3);

    multithreadGenerating(MAZE_SIZE * 4 / 5, 1);
    // TODO: Add check to see if okay maze
}

void Maze::moveNorth()
{
    for(int i = 0; i < MAZE_SIZE; i++)
    {
        // NOTE: The coordinates given is the top layer
        if(pathsNorth[i])
            currentPaths.push_back({i, MAZE_SIZE - 1});

        // This deletes any room that is being forgotten
        // NOTE it is 0 because it is getting rid of the bottom layer - which will become the top layer
        removeRoom(0, i);
    }
    changeYOffset(1);              // Changes the y offset because of the new layout
    if(currentPaths.size() > 0)    // This checks if there is any point to generate new paths
```

```cpp
        multithreadGenerating(LAYER_MAX_FOR_DIRECTIONS, 1);
    else
        updatePaths();
}

void Maze::moveSouth()
{   // These all do the same as moveNorth but just specialized for the direction
    for(int i = 0; i < MAZE_SIZE; i++)
    {
        if(pathsSouth[i])
            currentPaths.push_back({i, 0});

        removeRoom(MAZE_SIZE - 1, i);
    }
    changeYOffset(-1);
    if(currentPaths.size() > 0)
        multithreadGenerating(LAYER_MAX_FOR_DIRECTIONS, 1);
    else
        updatePaths();
}

void Maze::moveEast()
{
    for(int i = 0; i < MAZE_SIZE; i++)
    {
        if(pathsEast[i])
            currentPaths.push_back({MAZE_SIZE - 1, i});

        removeRoom(i, 0);
    }
    changeXOffset(1);
    if(currentPaths.size() > 0)
        multithreadGenerating(LAYER_MAX_FOR_DIRECTIONS, 1);
    else
        updatePaths();
}

void Maze::moveWest()
{
    for(int i = 0; i < MAZE_SIZE; i++)
    {
        if(pathsWest[i])
            currentPaths.push_back({0, i});

        removeRoom(i, MAZE_SIZE - 1);
    }
    changeXOffset(-1);
    if(currentPaths.size() > 0)
        multithreadGenerating(LAYER_MAX_FOR_DIRECTIONS, 1);
    else
        updatePaths();
}

Maze::EntranceState Maze::shouldBeOpen(Room *room, Direction nextEntrance, int prob, int *pathCount)
{
    if(room)
    {
        if(room->isOpen(nextEntrance))
        {   // If the opposite entrance is open then it needs to have the entrance open
            (*pathCount)++;
            return EntranceState::isOpen;
        }
        else    // If it is closed then there cannot be an entrance
            return EntranceState::isClosed;
    }
    else
    {   // Randomly generates whether the entrance will be open
        int r = Random::getNum(0, prob);
        if(r == 0)
        {
            (*pathCount)++;
            return EntranceState::isOpen;
        }
        return EntranceState::couldOpen;
    }
}

void Maze::forceEntrance(Maze::EntranceState *north, Maze::EntranceState *south, Maze::EntranceState *east, Maze::
    EntranceState *west)
{
    std::vector<EntranceState *> entrances;   // This will store the pointers to the entrance values
    entrances.reserve(3);

    // This checks if another entrance can be made - if so it adds it to the list
    if(!(*north))
        entrances.push_back(north);
    if(!(*south))
        entrances.push_back(south);
    if(!(*east))
        entrances.push_back(east);
    if(!(*west))
        entrances.push_back(west);

    if(entrances.size() == 1)   // This just makes sure that random
    {
        *entrances[0] = EntranceState::isOpen;
    }
    else if(entrances.size() > 0)
    {
        int r          = Random::getNum(0, entrances.size() - 1);
        *entrances[r] = EntranceState::isOpen;
    }
}

void Maze::updatePaths()
{
    /* NOTE: Probably don't even try to change where this is calculated! This must go here, because if it doesn't
    you would have to do something special with the paths variables or realise that you actually have to check at
        some point if the player is stuck.*/

    for(int i = 0; i < MAZE_SIZE; i++)
    {
        if(get(MAZE_SIZE - 1, i) && get(MAZE_SIZE - 1, i)->isOpen(Direction::north))
            pathsNorth[i] = true;
        if(get(0, i) && get(0, i)->isOpen(Direction::south))
            pathsSouth[i] = true;
        if(get(i, MAZE_SIZE - 1) && get(i, MAZE_SIZE - 1)->isOpen(Direction::east))
            pathsEast[i] = true;
        if(get(i, 0) && get(i, 0)->isOpen(Direction::west))
            pathsWest[i] = true;
    }
}

// !SECTION
```

../src/levels/Maze.cpp

3.8 Layers

layers/Layer.h

```
#pragma once

#include <vector>

#include "Event.h"
#include "Renderer.h"
#include "Shader.h"
#include "ShaderEffectsManager.h"

#include <memory>

class Layer
{
  protected:
    std::vector<uint16_t> m_ShaderEffectsIDs;

  public:
    Layer();
    virtual ~Layer();

    virtual void render()                                = 0;
    virtual void update()                                = 0;
    virtual bool eventCallback(const Event::Event &e)    = 0;

    virtual bool setEffect(Effect::Effect *e);

    std::vector<uint16_t> &getShaderEffects() { return m_ShaderEffectsIDs; }
#ifdef DEBUG
    virtual void imGuiRender();
#endif
};
```

../src/layers/Layer.h

layers/Layer.cpp

```
#include "Layer.h"

#include "Log.h"
#include "ShaderEffectsManager.h"

Layer::Layer() {}
Layer::~Layer() {}

bool Layer::setEffect(Effect::Effect *e)
{
    if(e->getType() == Effect::Effect::Type::shaderEffect)
    {
        Effect::ShaderEffectCarrier *ne = static_cast<Effect::ShaderEffectCarrier *>(e);
        m_ShaderEffectsIDs.push_back(ne->getID());
    }
    else if(e->getType() == Effect::Effect::Type::removeShaderEffect)
    {
        Effect::RemoveShaderEffect *ne = static_cast<Effect::RemoveShaderEffect *>(e);
        for(auto it = m_ShaderEffectsIDs.begin(); it != m_ShaderEffectsIDs.end();)
        {
            if(*it == ne->getID())
                it = m_ShaderEffectsIDs.erase(it);
            else
            {
                if(*it > ne->getID())
                    (*it)--;
                ++it;
            }
        }
    }
    return false;
}

void Layer::imGuiRender()
{
}
```

../src/layers/Layer.cpp

layers/MessageManager.h

```
#pragma once

#include "Text.h"

#include <chrono>

class MessageManager
{
  public:
    enum class Priority
    {
        High,
        Medium,
        Low
    };
    MessageManager(const MessageManager &) = delete;

    ~MessageManager();

    static MessageManager &get()
    {
        static MessageManager instance;
        return instance;
    }

    static void sendMessage(std::string message, Priority priority) { get().sendMessageImpl(message, priority); };
    static void update() { get().updateImpl(); };
    static void render() { get().renderImpl(); };

  private:
    struct Message
    {
        Text text;

        std::chrono::time_point<std::chrono::system_clock> timeCreated;
    };
    std::vector<Message> messages;

    MessageManager();

    void sendMessageImpl(std::string message, Priority priority);
    void updateImpl();
    void renderImpl();
};
```

../src/layers/MessageManager.h

layers/MessageManager.cpp

```
#include "MessageManager.h"

#include <ctime>

#include "Renderer.h"

#define TEXT_SCALE 60.0f

MessageManager::MessageManager()
{
    messages.reserve(10);
}

MessageManager::~MessageManager()
{
}

void MessageManager::sendMessageImpl(std::string message, Priority priority)
{
    glm::vec4 colour;
    switch(priority)
    {
    case Priority::High:
        colour = {1.0f, 0.0f, 0.0f, 1.0f};
        break;
    case Priority::Medium:
        colour = {0.992f, 0.737f, 0.18f, 1.0f};
        break;
    default:
        colour = {1.0f, 1.0f, 1.0f, 1.0f};
        break;
    }

    messages.push_back({{message, 0.0f, 0.0f, TEXT_SCALE, colour, false, true}, std::chrono::system_clock::now()})
    ;
}

void MessageManager::updateImpl()
{
    if(messages.size() != 0)
    {
        std::chrono::duration<double> elapsedSeconds = std::chrono::system_clock::now() - messages[0].timeCreated;
        if(elapsedSeconds.count() >= 5)
            messages.erase(messages.begin());
    }
}

void MessageManager::renderImpl()
{
    if(messages.size() != 0)
    {
        Vec2f pos = {200.0f, 200.0f};

        uint8_t layer       = 8;
        float   sizeOfLayer = TEXT_SCALE * (72.0f / 100.0f);

        float maxWidth = 40.0f;

        for(int i = messages.size() - 1; i > -1; i--)
        {
            messages[i].text.render(pos.x + 25.0f, pos.y + 25.0f + (messages.size() - 1 - i) * sizeOfLayer, layer)
    ;
            float width = Render::getTextWidth(messages[i].text.m_Text, TEXT_SCALE);
            if(width > maxWidth)
```

```
                maxWidth = width;
        }

        // NOTE: This will be rendered under the text because simple render happens before text render
        Render::rectangle(pos.x, pos.y, 0.0f, maxWidth + 50.0f, messages.size() * sizeOfLayer + 25.0f, {0.1f, 0.1f
    , 0.1f, 0.5f}, layer, false, true);
    }
}
```

../src/layers/MessageManager.cpp

3.9 Items

3.9.1 Base

item/Item.h

```
#pragma once

#include <string>

#include "Sprite.h"

class Item
{
  protected:
    Sprite::ID  m_SpriteID;
    std::string m_Name;

  public:
    Item();
    Item(std::string name, Sprite::ID spriteID);
    Item(const char *name, Sprite::ID spriteID);
    virtual ~Item();

    void render(float x, float y, double rotation, float size, uint8_t layer, bool isOverlay = false);

    Sprite::ID   getSpriteID();
    std::string *getName();
};
```

../src/item/Item.h

item/Item.cpp

```
#include "Item.h"

#include "Renderer.h"

Item::Item()
    : m_SpriteID(Sprite::ID::errorID), m_Name("I am an item")
{
}
Item::Item(std::string name, Sprite::ID spriteID)
    : m_SpriteID(spriteID), m_Name(name)
{
}

Item::Item(const char *name, Sprite::ID spriteID)
    : m_SpriteID(spriteID), m_Name(name)
{
}

Item::~Item()
{
}

void Item::render(float x, float y, double rotation, float size, uint8_t layer, bool isOverlay)
{
    Render::sprite(x, y, rotation, size, m_SpriteID, layer, isOverlay);
}

Sprite::ID   Item::getSpriteID() { return m_SpriteID; }
std::string *Item::getName() { return &m_Name; }
```

../src/item/Item.cpp

3.9.2 Potions

item/potions/Potion.h

```
#pragma once

#include "Item.h"
#include "Mob.h"

#include <functional>

class Potion : public Item
{
  public:
    enum class Type
    {
        Health,
        HealthRegen,
        HealthMagic,
        HealthHuge,

        Stamina,
        StaminaRegen,
        StaminaMagic,
        StaminaHuge,

        Book,
        MagicBook,
        Food
    };

  private:
    std::function<void(Mob *)> m_EffectFunc;

  public:
    Potion(Type type);
    Potion(const char *name, Sprite::ID spriteID, std::function<void(Mob *)> effect);
    virtual ~Potion() override;

    void useOn(Mob *mob);
};
```

../src/item/potions/Potion.h

item/potions/Potion.cpp

```
#include "Potion.h"

#include "RandomGen.h"

Potion::Potion(Type type)
    : Item(std::string(), Sprite::ID::errorID)
{
    // TODO: ADD REGEN!
    switch(type)
    {
    case Type::Health:
        m_Name       = "Health Potion";
        m_SpriteID   = Sprite::ID::potionRed;
        m_EffectFunc = [](Mob *mob) {
            mob->changeHealth(25);
        };

        break;
    case Type::HealthRegen:
        m_Name       = "Health Regen Potion";
        m_SpriteID   = Sprite::ID::potionRedRegen;
        m_EffectFunc = [](Mob *mob) {
            mob->changeHealth(25);
        };

        break;
    case Type::HealthMagic:
        m_Name       = "Magic Health Potion";
        m_SpriteID   = Sprite::ID::potionRedMagic;
        m_EffectFunc = [](Mob *mob) {
            mob->changeHealth(50);
        };

        break;
    case Type::HealthHuge:
        m_Name       = "Huge Health Potion";
        m_SpriteID   = Sprite::ID::potionRedHuge;
        m_EffectFunc = [](Mob *mob) {
            mob->changeHealth(75);
        };

        break;
    case Type::Stamina:
        m_Name       = "Stamina Potion";
        m_SpriteID   = Sprite::ID::potionGreen;
        m_EffectFunc = [](Mob *mob) {
            mob->changeStamina(25);
        };

        break;
    case Type::StaminaRegen:
        m_Name       = "Stamina Regen Potion";
        m_SpriteID   = Sprite::ID::potionGreenRegen;
        m_EffectFunc = [](Mob *mob) {
            mob->changeStamina(25);
        };

        break;
    case Type::StaminaMagic:
        m_Name       = "Magic Stamina Potion";
        m_SpriteID   = Sprite::ID::potionGreenMagic;
        m_EffectFunc = [](Mob *mob) {
            mob->changeStamina(50);
        };

        break;
    case Type::StaminaHuge:
        m_Name       = "Huge Stamina Potion";
        m_SpriteID   = Sprite::ID::potionGreenHuge;
        m_EffectFunc = [](Mob *mob) {
            mob->changeStamina(75);
        };

        break;
    case Type::Book:
        m_Name       = "Book";
        m_SpriteID   = SPRITE_BOOK_START + Random::getNum(0, SPRITE_BOOK_NUM - 1);
        m_EffectFunc = [](Mob *mob) {
            mob->changeBoredom(25);
        };

        break;
    case Type::MagicBook:
        m_Name       = "Magic Book";
        m_SpriteID   = SPRITE_MAGIC_BOOK_START + Random::getNum(0, SPRITE_MAGIC_BOOK_NUM - 1);
        m_EffectFunc = [](Mob *mob) {
            mob->changeBoredom(50);
        };

        break;
    case Type::Food:
        m_SpriteID   = SPRITE_FOOD_START + Random::getNum(0, SPRITE_FOOD_NUM - 1);
        m_Name       = "Food";
        m_EffectFunc = [](Mob *mob) {
            mob->changeHealth(Random::getNum(5, 15));
        };

        break;
    default:
        Log::warning("Unknown potion!");
        m_Name       = "Error";
        m_EffectFunc = [](Mob *mob) {};
        break;
    }
}

Potion::Potion(const char *name, Sprite::ID spriteID, std::function<void(Mob *)> effect)
    : Item(name, spriteID), m_EffectFunc(effect)
{
}

Potion::~Potion()
{
}

void Potion::useOn(Mob *mob)
```

```
{
    m_EffectFunc(mob);
}
```

../src/item/potions/Potion.cpp

3.9.3 Weapons

item/weapons/Weapon.h

```
#pragma once

#include "Item.h"

#include "Entity.h"
#include "Event.h"

class Level;

class Weapon : public Item
{
  protected:
    float m_Damage;
    uint16_t m_Cooldown;
    uint16_t m_CooldownMax;

  public:
    Weapon(float damage);
    Weapon(const char *name, float damage, Sprite::ID spriteID);
    Weapon(const char *name, float damage, uint16_t cooldown, Sprite::ID spriteID);
    virtual ~Weapon() override;

    virtual void update();

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) = 0;

    bool canUse() { return m_Cooldown == 0; }
};
```

../src/item/weapons/Weapon.h

item/weapons/Weapon.cpp

```
#include "Weapon.h"

Weapon::Weapon(float damage)
    : Item(), m_Damage(damage), m_Cooldown(0), m_CooldownMax(0)
{
}

Weapon::Weapon(const char *name, float damage, Sprite::ID spriteID)
    : Item(name, spriteID), m_Damage(damage), m_Cooldown(0), m_CooldownMax(0)
{
}

Weapon::Weapon(const char *name, float damage, uint16_t cooldown, Sprite::ID spriteID)
    : Item(name, spriteID), m_Damage(damage), m_Cooldown(0), m_CooldownMax(cooldown)
{
}

Weapon::~Weapon() {}

void Weapon::update()
{
    if(m_Cooldown != 0)
        m_Cooldown--;
}
```

../src/item/weapons/Weapon.cpp

item/weapons/general/Boomerang.h

```
#pragma once

#include "Weapon.h"

#include "Projectile.h"
#include "Application.h"

class Boomerang : public Weapon
{
  public:
    Boomerang();
    virtual ~Boomerang() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/general/Boomerang.h

item/weapons/general/Boomerang.cpp

```
#include "Boomerang.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

Boomerang::Boomerang()
    : Weapon("Boomerang", 20.0f, 50, Sprite::ID::weaponBoomerang) {}
Boomerang::~Boomerang() {}

void Boomerang::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::
    Boomerang));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/general/Boomerang.cpp

item/weapons/general/Bow.h

```
#pragma once

#include "Weapon.h"

#include "Projectile.h"
#include "Application.h"

class Bow : public Weapon
{
  public:
    Bow();
    virtual ~Bow() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/general/Bow.h

item/weapons/general/Bow.cpp

```
#include "Bow.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

Bow::Bow()
    : Weapon("Bow", 15.0f, 30, Sprite::ID::weaponBow) {}
Bow::~Bow() {}

void Bow::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Arrow));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/general/Bow.cpp

item/weapons/general/Crossbow.h

```
#pragma once

#include "Weapon.h"

#include "Projectile.h"
#include "Application.h"

class Crossbow : public Weapon
{
  public:
    Crossbow();
    virtual ~Crossbow() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/general/Crossbow.h

item/weapons/general/Crossbow.cpp

```
#include "Crossbow.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

Crossbow::Crossbow()
    : Weapon("Crossbow", 20.0f, 25, Sprite::ID::weaponCrossbow) {}
Crossbow::~Crossbow() {}

void Crossbow::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Arrow));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/general/Crossbow.cpp

item/weapons/general/Sling.h

```
#pragma once

#include "Weapon.h"
```

```
#include "Application.h"
#include "Projectile.h"

class Sling : public Weapon
{
  public:
    Sling();
    virtual ~Sling() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/general/Sling.h

item/weapons/general/Sling.cpp

```
#include "Sling.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

Sling::Sling()
    : Weapon("Slingshot", 8.0f, 15, Sprite::ID::weaponSling) {}
Sling::~Sling() {}

void Sling::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Rock));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/general/Sling.cpp

item/weapons/staffs/DarkStaff.h

```
#pragma once

#include "Weapon.h"

#include "Projectile.h"
#include "Application.h"

class DarkStaff : public Weapon
{
  public:
    DarkStaff();
    virtual ~DarkStaff() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/staffs/DarkStaff.h

item/weapons/staffs/DarkStaff.cpp

```
#include "DarkStaff.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

DarkStaff::DarkStaff()
    : Weapon("Dark Staff", 18.0f, 15, Sprite::ID::weaponDarkStaff) {}
```

```
DarkStaff::~DarkStaff() {}

void DarkStaff::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Dark));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/staffs/DarkStaff.cpp

item/weapons/staffs/EarthStaff.h

```
#pragma once

#include "Weapon.h"

#include "Application.h"
#include "Projectile.h"

class EarthStaff : public Weapon
{
  public:
    EarthStaff();
    virtual ~EarthStaff() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/staffs/EarthStaff.h

item/weapons/staffs/EarthStaff.cpp

```
#include "EarthStaff.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

EarthStaff::EarthStaff()
    : Weapon("Earth Staff", 15.0f, 15, Sprite::ID::weaponEarthStaff) {}
EarthStaff::~EarthStaff() {}

void EarthStaff::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Nature))
    ;
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/staffs/EarthStaff.cpp

item/weapons/staffs/FireStaff.h

```cpp
#pragma once

#include "Weapon.h"

#include "Projectile.h"
#include "Application.h"

class FireStaff : public Weapon
{
  public:
    FireStaff();
    virtual ~FireStaff() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/staffs/FireStaff.h

item/weapons/staffs/FireStaff.cpp

```cpp
#include "FireStaff.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

FireStaff::FireStaff()
    : Weapon("Fire Staff", 10.0f, 20, Sprite::ID::weaponFireStaff) {}
FireStaff::~FireStaff() {}

void FireStaff::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Fire));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/staffs/FireStaff.cpp

item/weapons/staffs/FrostStaff.h

```cpp
#pragma once

#include "Weapon.h"

#include "Application.h"
#include "Projectile.h"

class FrostStaff : public Weapon
{
  public:
    FrostStaff();
    virtual ~FrostStaff() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/staffs/FrostStaff.h

item/weapons/staffs/FrostStaff.cpp

```cpp
#include "FrostStaff.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
```

```
#include "ParticleSpawner.h"

FrostStaff::FrostStaff()
    : Weapon("Frost Staff", 15.0f, 30, Sprite::ID::weaponFrostStaff) {}
FrostStaff::~FrostStaff() {}

void FrostStaff::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Frost));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/staffs/FrostStaff.cpp

item/weapons/staffs/GoldStaff.h

```
#pragma once

#include "Weapon.h"

#include "Application.h"
#include "Projectile.h"

class GoldStaff : public Weapon
{
  public:
    GoldStaff();
    virtual ~GoldStaff() override;

    virtual void attack(Level *level, Entity &e, Direction dir, bool hold) override;
};
```

../src/item/weapons/staffs/GoldStaff.h

item/weapons/staffs/GoldStaff.cpp

```
#include "GoldStaff.h"

#include "KeyDefinitions.h"

#include "Level.h"
#include "Mob.h"
#include "ParticleSpawner.h"

GoldStaff::GoldStaff()
    : Weapon("Gold Staff", 30.0f, 25, Sprite::ID::weaponGoldStaff) {}
GoldStaff::~GoldStaff() {}

void GoldStaff::attack(Level *level, Entity &e, Direction dir, bool hold)
{
    Mob *m = dynamic_cast<Mob *>(&e);
    if(m_Cooldown == 0)
    {
        float damage;
        if(m)
            damage = m->getDamage(m_Damage / 2.0f, m_Damage);
        else
            damage = 0.0f;

        level->addProjectile(new Projectile(e.getX(), e.getY(), damage, dir, &e, level, Projectile::Type::Gold));
        if(m)
        {
            m_Cooldown = m->getWeaponDelay(m_CooldownMax);
            m->hasUsedWeapon();
        }
    }
}
```

../src/item/weapons/staffs/GoldStaff.cpp

3.10 GUI

3.10.1 Layer

gui/GUILayer.h

```
#pragma once

#include "Layer.h"

#include <memory>

#include "Level.h"
#include "MenuObject.h"
#include "TransferObject.h"

class GUILayer : public Layer
{
  public:
    enum class Type
    {
        MainMenu,
        GameOverlay,
        PlayerInventory,
        ChestInventory,
        NPCInventory,
        NPCInteraction,
        ExitMenu,
        PlayerDeath,
        PlayerWin,
    };

  private:
    Level *                   m_ConnectedLevel;
    std::vector<MenuObject *> m_Objects;

  public:
    GUILayer();
    GUILayer(Level *connectedLevel);
    GUILayer(Type genType, Level *connectedLevel);
    virtual ~GUILayer() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;

    void addMenuObject(MenuObject *object);

    void setConnectedLevel(Level *level);

    void transferObject(TransferObject *obj);

    std::vector<MenuObject *> &getObjects() { return m_Objects; }
    Level *                    getConnectedLevel() { return m_ConnectedLevel; }

#ifdef DEBUG
    virtual void imGuiRender() override
    {
    }
#endif
};
```

../src/gui/GUILayer.h

gui/GUILayer.cpp

```
#include "GUILayer.h"

#include "Button.h"
#include "GUILayer.h"
#include "MenuBackgroundObject.h"
#include "MenuItemHolderManager.h"
#include "StatBar.h"
#include "TextMenuObject.h"
#include "TransferObject.h"
#include "Image.h"

#include "Sprite.h"

#include "Utils.h"

#include "Application.h"
```

```
GUILayer::GUILayer()
    : m_ConnectedLevel(nullptr)
{
}

GUILayer::GUILayer(Level *connectedLevel)
    : m_ConnectedLevel(connectedLevel)
{
}

GUILayer::GUILayer(GUILayer::Type genType, Level *connectedLevel)
    : m_ConnectedLevel(connectedLevel)
{
    switch(genType)
    {
    case GUILayer::Type::MainMenu: // TODO: Add scale
    {
        auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
            *width  = Application::getWidth();
            *height = Application::getHeight();
            *x      = Application::getWidth() / 2;
            *y      = Application::getHeight() / 2;
        };

        addMenuObject(new MenuBackground(backgroundPosFunc, this, {1.0f, 1.0f, 1.0f, 1.0f}, []() {}));

        auto posFunc = [](float *x, float *y, float *width, float *height) {
            *width  = Application::getWidth() / 4;
            *height = 1080.0f * (*width) / 1920.0f;
            *x      = Application::getWidth() / 2;
            *y = 3 * Application::getHeight() / 4;
        };

        addMenuObject(new Image(posFunc, Sprite::ID::menuTitle, this));

        auto exitFunc = []() {
            Application::closeApplication();
        };

        auto exitPos = [](float *x, float *y, float *width, float *height) {
            *width  = 400;
            *height = 100;
            *x      = Application::getWidth() / 2;
            *y      = Application::getHeight() / 2 - (*height);
        };

        Text exitText("Exit");

        addMenuObject(new Button(exitText, exitPos, this, exitFunc));

        auto startFunc = []() {
            Application::startGame();
        };

        auto startPos = [](float *x, float *y, float *width, float *height) {
            *width  = 400;
            *height = 100;
            *x      = Application::getWidth() / 2;
            *y      = Application::getHeight() / 2 + (*height);
        };

        Text startText("Start");

        addMenuObject(new Button(startText, startPos, this, startFunc));

        break;
    }

    case GUILayer::Type::GameOverlay:   // TODO: Turn these into functions?
    {
        auto clickedFunc = [this](int index, Level *level) {
            m_ConnectedLevel->getPlayer()->setCurrentWeapon(index);
        };
        addMenuObject(new MIHManager(0, 0, 300, 100, 100, this, m_ConnectedLevel->getPlayer()->getWeapons(),
    clickedFunc, m_ConnectedLevel->getPlayer()->getCurrentWeaponPointer()));

        auto posFunc = [](float *x, float *y, float *width, float *height) {
            *x      = Application::getWidth() / 2;
            *y      = 20;
        *width  = Application::getWidth() / 3;
        *height = 10;
    };
    addMenuObject(new StatBar(posFunc, this, m_ConnectedLevel->getPlayer()->getHealthPointer(),
m_ConnectedLevel->getPlayer()->getMaxHealthPointer()));

    auto exitFunc = []() {
        Event::ChangeGUIActiveLayer e(InGameGUILayer::exitMenu);
        Application::callEvent(e, true);
    };

    auto exitPos = [](float *x, float *y, float *width, float *height) {
        *width  = 200;
        *height = 50;
        *x      = Application::getWidth() - (*width) / 2;
        *y      = Application::getHeight() - (*height) / 2;
    };

    Text exitText("Exit");

    addMenuObject(new Button(exitText, exitPos, this, exitFunc));

    break;
}

case GUILayer::Type::PlayerInventory:
{
    TransferObject *transfer = new TransferObject(TILE_SIZE, TILE_SIZE, this);
    addMenuObject(transfer);

    auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 550;
        *height = 575;
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2 + 75.0f / 2.0f;
    };

    auto exitFunc = []() {
        Event::ChangeGUIActiveLayer e(InGameGUILayer::overlay);
        Application::callEvent(e, true);
    };

    addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.3f, 0.3f, 0.3f, 0.9f}, exitFunc));

    auto clickedFunc = [this](int index, Level *level) {   // TODO: remove the level *
        m_ConnectedLevel->getPlayer()->useItemInInventory(index);
    };

    auto posFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 500;
        *height = 400;
        *x      = Application::getWidth() / 2 - *width / 2;
        *y      = Application::getHeight() / 2 + *height / 2 - 112.5f;
    };
    addMenuObject(new MIHManager(posFunc, 100, this, m_ConnectedLevel->getPlayer()->getInventory(),
clickedFunc));

    auto clickedWeaponFunc = [this](int index, Level *level) {
        m_ConnectedLevel->getPlayer()->setCurrentWeapon(index);
    };
    auto posWeaponFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 300;
        *height = 100;
        *x      = Application::getWidth() / 2 - *width / 2;
        *y      = Application::getHeight() / 2 + 200;
    };
    addMenuObject(new MIHManager(posWeaponFunc, 100, this, m_ConnectedLevel->getPlayer()->getWeapons(),
clickedWeaponFunc, m_ConnectedLevel->getPlayer()->getCurrentWeaponPointer()));

    break;
}

case GUILayer::Type::ChestInventory:
{
    TransferObject *transfer = new TransferObject(TILE_SIZE, TILE_SIZE, this);
    addMenuObject(transfer);

    auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 900;
        *height = 675;
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2 + 75.0f / 2.0f;
    };
    auto exitFunc = []() {
        Event::ChangeGUIActiveLayer e(InGameGUILayer::overlay);
        Application::callEvent(e, true);
    };
    addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.3f, 0.3f, 0.3f, 0.9f}, exitFunc));

    auto clickedFunc = [](int index, Level *level) {   // TODO: remove the level * and do something here
    };

    auto posFuncForPlayerInventory = [](float *x, float *y, float *width, float *height) {
        *width  = 400;
        *height = 500;
        *x      = Application::getWidth() / 2 - *width - 10.0f;
        *y      = Application::getHeight() / 2 + *height / 2 - 112.5f;
    };
    addMenuObject(new MIHManager(posFuncForPlayerInventory, 100, this, m_ConnectedLevel->getPlayer()->
getInventory(), clickedFunc));

    auto posFuncForChestInventory = [](float *x, float *y, float *width, float *height) {
        *width  = 400;
        *height = 500;
        *x      = Application::getWidth() / 2 + 10.0f;
        *y      = Application::getHeight() / 2 + *height / 2 - 112.5f;
    };
    MIHManager *chestSection = new MIHManager(posFuncForChestInventory, 100, this, nullptr, clickedFunc,
nullptr, GUIInventoryIDCode::inventory);
    addMenuObject(chestSection);

    auto clickedWeaponFunc = [this](int index, Level *level) {
        m_ConnectedLevel->getPlayer()->setCurrentWeapon(index);
    };
    auto posWeaponFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 300;
        *height = 100;
        *x      = Application::getWidth() / 2 - *width / 2;
        *y      = Application::getHeight() / 2 + 250;
    };
    addMenuObject(new MIHManager(posWeaponFunc, 100, this, m_ConnectedLevel->getPlayer()->getWeapons(),
clickedWeaponFunc, m_ConnectedLevel->getPlayer()->getCurrentWeaponPointer()));

    break;
}

case GUILayer::Type::NPCInventory:
{
    TransferObject *transfer = new TransferObject(TILE_SIZE, TILE_SIZE, this);
    addMenuObject(transfer);

    auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 900;
        *height = 675;
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2 + 75.0f / 2.0f;
    };
    auto exitFunc = []() {
        Event::ChangeGUIActiveLayer e(InGameGUILayer::overlay);
        Application::callEvent(e, true);
    };
    addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.3f, 0.3f, 0.3f, 0.9f}, exitFunc));

    auto clickedFunc = [](int index, Level *level) {   // TODO: remove the level * and do something here
    };

    auto posFuncForPlayerInventory = [](float *x, float *y, float *width, float *height) {
        *width  = 400;
        *height = 500;
        *x      = Application::getWidth() / 2 - *width - 10.0f;
        *y      = Application::getHeight() / 2 + *height / 2 - 112.5f;
    };
    addMenuObject(new MIHManager(posFuncForPlayerInventory, 100, this, m_ConnectedLevel->getPlayer()->
getInventory(), clickedFunc));

    auto posFuncForChestInventory = [](float *x, float *y, float *width, float *height) {
        *width  = 400;
        *height = 500;
        *x      = Application::getWidth() / 2 + 10.0f;
        *y      = Application::getHeight() / 2 + *height / 2 - 112.5f;
    };
    MIHManager *chestSection = new MIHManager(posFuncForChestInventory, 100, this, nullptr, clickedFunc,
nullptr, GUIInventoryIDCode::inventory);
    addMenuObject(chestSection);

    auto playerClickedWeaponFunc = [this](int index, Level *level) {
        m_ConnectedLevel->getPlayer()->setCurrentWeapon(index);
    };
    auto playerPosWeaponFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 300;
        *height = 100;
        *x      = Application::getWidth() / 2 - *width / 2 - 210.0f;
        *y      = Application::getHeight() / 2 + 250;
    };
    addMenuObject(new MIHManager(playerPosWeaponFunc, 100, this, m_ConnectedLevel->getPlayer()->getWeapons(),
playerClickedWeaponFunc, m_ConnectedLevel->getPlayer()->getCurrentWeaponPointer()));

    auto npcClickedWeaponFunc = [this](int index, Level *level) {
    };
    auto npcPosWeaponFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 300;
        *height = 100;
        *x      = Application::getWidth() / 2 - *width / 2 + 210.0f;
        *y      = Application::getHeight() / 2 + 250;
    };
    addMenuObject(new MIHManager(npcPosWeaponFunc, 100, this, nullptr, npcClickedWeaponFunc, nullptr,
GUIInventoryIDCode::weapons));

    break;
}

case GUILayer::Type::NPCInteraction:
{
    auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 470;
        *height = 100;
        *x      = Application::getWidth() / 2;
        *y      = (*height) / 2 + 50.0f;
    };

    auto exitFunc = []() {
        Event::PlayerResponse e(Event::PlayerResponse::Response::reject);
        Application::callEvent(e);
    };

    addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.3f, 0.3f, 0.3f, 0.9f}, exitFunc));

    auto rejectFunc = []() {
        Event::PlayerResponse e(Event::PlayerResponse::Response::reject);
        Application::callEvent(e);
    };

    auto rejectPos = [](float *x, float *y, float *width, float *height) {
        *width  = 200;
        *height = 50;
        *x      = Application::getWidth() / 2 - (*width) / 2 - 10.0f;
        *y      = 100.0f;
    };

    Text rejectText("Reject");

    addMenuObject(new Button(rejectText, rejectPos, this, rejectFunc));

    auto acceptFunc = []() {
        Event::PlayerResponse e(Event::PlayerResponse::Response::accept);
        Application::callEvent(e);
    };

    auto acceptPos = [](float *x, float *y, float *width, float *height) {
        *width  = 200;
        *height = 50;
        *x      = Application::getWidth() / 2 + (*width) / 2 + 10.0f;
        *y      = 100.0f;
    };

    Text acceptText("Accept");

    addMenuObject(new Button(acceptText, acceptPos, this, acceptFunc));

    break;
```

```
}

case GUILayer::Type::ExitMenu:
{
    auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
        *width  = 250;
        *height = 175;
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2;
    };

    auto exitFunc = []() {
        Event::ChangeGUIActiveLayer e(InGameGUILayer::overlay);
        Application::callEvent(e, true);
    };

    addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.3f, 0.3f, 0.3f, 0.9f}, exitFunc));

    auto exitGameFunc = []() {
        Application::exitGame();
    };

    auto exitPos = [](float *x, float *y, float *width, float *height) {
        *width  = 200;
        *height = 50;
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2 - (*height) / 2 - 12.5f;
    };

    Text exitText("Main Menu");

    addMenuObject(new Button(exitText, exitPos, this, exitGameFunc));

    auto continuePos = [](float *x, float *y, float *width, float *height) {
        *width  = 200;
        *height = 50;
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2 + (*height) / 2 + 12.5f;
    };

    Text continueText("Continue");

    addMenuObject(new Button(continueText, continuePos, this, exitFunc));

    break;
}

case GUILayer::Type::PlayerDeath:
{
    auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
        *width  = Application::getWidth();
        *height = Application::getHeight();
        *x      = Application::getWidth() / 2;
        *y      = Application::getHeight() / 2;
    };

    auto exitFunc = []() {
        Application::exitGame();
    };

    addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.0f, 0.0f, 0.0f, 1.0f}, exitFunc));

    auto exitPos = [](float *x, float *y, float *width, float *height) {
        *width  = 600;
        *height = 200;
        *x      = Application::getWidth() / 2;
        *y      = (*height) / 2 + 200.0f;
    };

    Text exitText("Main Menu", 0.0f, 0.0f, 150.0f, {1.0f, 1.0f, 1.0f, 1.0f});

    addMenuObject(new Button(exitText, exitPos, this, exitFunc));

    auto textPos = [](float *x, float *y) {
        *x = Application::getWidth() / 2;
        *y = Application::getHeight() / 2 + 150.0f;
    };

    Text text("YOU DIED...", 0.0f, 0.0f, 800.0f, {1.0f, 0.0f, 0.0f, 1.0f});
```

```
        addMenuObject(new TextMenuObject(text, textPos, this));

        break;
    }

    case GUILayer::Type::PlayerWin:
    {
        auto backgroundPosFunc = [](float *x, float *y, float *width, float *height) {
            *width  = Application::getWidth();
            *height = Application::getHeight();
            *x      = Application::getWidth() / 2;
            *y      = Application::getHeight() / 2;
        };

        auto exitFunc = []() {
            Application::exitGame();
        };

        addMenuObject(new MenuBackground(backgroundPosFunc, this, {0.0f, 0.0f, 0.0f, 1.0f}, exitFunc));

        auto exitPos = [](float *x, float *y, float *width, float *height) {
            *width  = 600;
            *height = 200;
            *x      = Application::getWidth() / 2;
            *y      = (*height) / 2 + 200.0f;
        };

        Text exitText("Main Menu", 0.0f, 0.0f, 150.0f, {1.0f, 1.0f, 1.0f, 1.0f});

        addMenuObject(new Button(exitText, exitPos, this, exitFunc));

        auto textPos = [](float *x, float *y) {
            *x = Application::getWidth() / 2;
            *y = Application::getHeight() / 2 + 150.0f;
        };

        Text text("YOU WIN!", 0.0f, 0.0f, 800.0f, {1.0f, 0.843f, 0.0f, 1.0f});

        addMenuObject(new TextMenuObject(text, textPos, this));

        break;
    }

    default:
        Log::warning("Unknown GUI type");
        break;
    }
}

GUILayer::~GUILayer()
{
    for(MenuObject *obj : m_Objects)
        delete obj;
}

void GUILayer::render()
{
    for(MenuObject *obj : m_Objects)
        obj->render();
}

void GUILayer::update()
{
    for(MenuObject *obj : m_Objects)
        obj->update();
}

bool GUILayer::eventCallback(const Event::Event &e)
{
    for(MenuObject *obj : m_Objects)
    {
        if(obj->eventCallback(e))
            return true;
    }
    return false;
}

void GUILayer::addMenuObject(MenuObject *object)
{
    m_Objects.push_back(object);
```

```
}

void GUILayer::setConnectedLevel(Level *level)
{
    m_ConnectedLevel = level;
}

void GUILayer::transferObject(TransferObject *transfer)
{
    for(MenuObject *obj : m_Objects)
    {
        MIHManager *manager = dynamic_cast<MIHManager *>(obj);
        if(manager)
        {
            int hoverBlock = manager->getIndexMouseAt();
            if(hoverBlock != -1)
            {
                manager->transferItem(transfer);
                break;
            }
        }
    }
}
```

../src/gui/GUILayer.cpp

gui/GUIStack.h

```
#pragma once

#include "Layer.h"

#include <vector>

#include "GUILayer.h"
#include "Level.h"

class GUIStack : public Layer
{
  public:
  private:
    int m_ActiveLayer;

    std::vector<GUILayer *> m_Layers;

  public:
    GUIStack(Level *level);
    virtual ~GUIStack() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
};
```

../src/gui/GUIStack.h

gui/GUIStack.cpp

```
#include "GUIStack.h"

#include "Application.h"

GUIStack::GUIStack(Level *level)
    : m_ActiveLayer(0)
{
    m_Layers.push_back(new GUILayer(GUILayer::Type::GameOverlay, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::PlayerInventory, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::ChestInventory, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::NPCInventory, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::NPCInteraction, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::ExitMenu, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::PlayerDeath, level));
    m_Layers.push_back(new GUILayer(GUILayer::Type::PlayerWin, level));
}

GUIStack::~GUIStack()
{
    for(GUILayer *layer : m_Layers)
        delete layer;
}

void GUIStack::render()
```

```
{
    if(m_ActiveLayer != -1)
        m_Layers[m_ActiveLayer]->render();
}

void GUIStack::update()
{
    if(m_ActiveLayer != -1)
        m_Layers[m_ActiveLayer]->update();
}

bool GUIStack::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::changeGUILayer)
    {
        const Event::ChangeGUIActiveLayer &ne = static_cast<const Event::ChangeGUIActiveLayer &>(e);

        m_ActiveLayer = static_cast<int>(ne.layer);

        if(ne.layer == InGameGUILayer::overlay)
            Application::setIsPaused(false);
        else
            Application::setIsPaused(true);

        return true;
    }
    else if(e.getType() == Event::EventType::windowResize)
    {
        for(GUILayer *layer : m_Layers)
            layer->eventCallback(e);
    }
    else
        return m_Layers[m_ActiveLayer]->eventCallback(e);

    return false;
}
```

../src/gui/GUIStack.cpp

3.10.2 Objects

gui/objects/MenuObject.h

```
#pragma once

#include <functional>

#include "Event.h"
#include "Layer.h"

class MenuObject
{
  protected:
    float x, y;
    float                                                   width, height;
    std::function<void(float *, float *, float *, float *)> positionFunc;

    Layer *m_Layer;

  public:
    MenuObject(float x, float y, float width, float height, Layer *layer);
    MenuObject(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer);
    virtual ~MenuObject();

    virtual void render() = 0;
    virtual void update() = 0;
    virtual bool eventCallback(const Event::Event &e);

    float getX() { return x; }
    float getY() { return y; }
    Layer *getLayer() { return m_Layer; }
};
```

../src/gui/objects/MenuObject.h

gui/objects/MenuObject.cpp

```
#include "MenuObject.h"

#include "GUILayer.h"
```

```
#include "Log.h"

MenuObject::MenuObject(float x, float y, float width, float height, Layer *layer)
    : x(x), y(y), width(width), height(height), positionFunc([](float *, float *, float *, float *) {}), m_Layer(
    layer)
{
    GUILayer *guilayer = dynamic_cast<GUILayer *>(layer);
    if(!guilayer)
        Log::critical("Wrong layer type given!", LOGINFO);
}

MenuObject::MenuObject(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer)
    : positionFunc(posFunc), m_Layer(layer)
{
    positionFunc(&x, &y, &width, &height);
}

MenuObject::~MenuObject()
{
}

bool MenuObject::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::windowResize)
    {
        positionFunc(&x, &y, &width, &height);
    }
    return false;
}
```

../src/gui/objects/MenuObject.cpp

gui/objects/Button.h

```
#pragma once

#include "MenuObject.h"

#include <GLM.h>
#include <functional>
#include <string>

#include "Text.h"

class Button : public MenuObject
{
  public:
    enum State
    {
        None = 0,
        Hover,
        Press,
    };

  protected:
    Text m_Text;

    glm::vec4 m_BackgroundColour;
    glm::vec4 m_HoverColour;
    glm::vec4 m_PressColour;
    glm::vec4 m_BorderColour;

    uint16_t pressCalldown;

    State m_State;

    std::function<void()> buttonPressFunc;

  public:
    Button(Text text, float x, float y, float width, float height, Layer *layer, std::function<void()>
    buttonPressFunc);
    Button(Text text, std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, std::function
    <void()> buttonPressFunc);
    Button(Text text, float x, float y, float width, float height, Layer *layer, glm::vec4 backgroundColour, glm::
    vec4 hoverColour, glm::vec4 pressColour, std::function<void()> buttonPressFunc);
    Button(Text text, std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, glm::vec4
    backgroundColour, glm::vec4 hoverColour, glm::vec4 pressColour, std::function<void()> buttonPressFunc);
    virtual ~Button() override;

    virtual void update() override;
    virtual void render() override;
    virtual bool eventCallback(const Event::Event &e) override;
};
```

../src/gui/objects/Button.h

gui/objects/Button.cpp

```
#include "Button.h"

#include "Renderer.h"

Button::Button(Text text, float x, float y, float width, float height, Layer *layer, std::function<void()>
    buttonPressFunc)
    : MenuObject(x, y, width, height, layer),
      m_Text(text),
      m_BackgroundColour({0.3f, 0.3f, 0.3f, 1.0f}),
      m_HoverColour({0.2f, 0.2f, 0.2f, 1.0f}),
      m_PressColour({0.0f, 0.0f, 0.0f, 1.0f}),
      m_BorderColour({0.0f, 0.0f, 0.0f, 1.0f}),
      pressCalldown(0),
      m_State(State::None),
      buttonPressFunc(buttonPressFunc)
{
}

Button::Button(Text text, std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, std::
    function<void()> buttonPressFunc)
    : MenuObject(posFunc, layer),
      m_Text(text),
      m_BackgroundColour({0.3f, 0.3f, 0.3f, 1.0f}),
      m_HoverColour({0.2f, 0.2f, 0.2f, 1.0f}),
      m_PressColour({0.0f, 0.0f, 0.0f, 1.0f}),
      m_BorderColour({0.0f, 0.0f, 0.0f, 1.0f}),
      pressCalldown(0),
      m_State(State::None),
      buttonPressFunc(buttonPressFunc)
{
}

Button::Button(Text text, float x, float y, float width, float height, Layer *layer, glm::vec4 backgroundColour,
    glm::vec4 hoverColour, glm::vec4 pressColour, std::function<void()> buttonPressFunc)
    : MenuObject(x, y, width, height, layer),
      m_Text(text),
      m_BackgroundColour(backgroundColour),
      m_HoverColour(hoverColour),
      m_PressColour(pressColour),
      m_BorderColour({0.0f, 0.0f, 0.0f, 1.0f}),
      pressCalldown(0),
      m_State(State::None),
      buttonPressFunc(buttonPressFunc)
{
}

Button::Button(Text text, std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, glm::vec4
     backgroundColour, glm::vec4 hoverColour, glm::vec4 pressColour, std::function<void()> buttonPressFunc)
    : MenuObject(posFunc, layer),
      m_Text(text),
      m_BackgroundColour(backgroundColour),
      m_HoverColour(hoverColour),
      m_PressColour(pressColour),
      m_BorderColour({0.0f, 0.0f, 0.0f, 1.0f}),
      pressCalldown(0),
      m_State(State::None),
      buttonPressFunc(buttonPressFunc)
{
}

Button::~Button()
{
}

void Button::update()
{
    if(pressCalldown > 0)
        pressCalldown--;
    else
    {
        Vec2f mousePos = Event::getMousePos();
        if(mousePos.x > x - width / 2 && mousePos.x < x + width / 2 && mousePos.y > y - height / 2 && mousePos.y <
     y + height / 2)
            m_State = State::Hover;
        else
            m_State = State::None;
    }
}

void Button::render()
{
    uint8_t layer = 7;
    if(m_State == State::None)
        Render::rectangle(x, y, width, height, m_BackgroundColour, 2.0f, m_BorderColour, layer, true, true);
    else if(m_State == State::Hover)
        Render::rectangle(x, y, width, height, m_HoverColour, 2.0f, m_BorderColour, layer, true, true);
    else
        Render::rectangle(x, y, width, height, m_PressColour, 2.0f, m_BorderColour, layer, true, true);

    m_Text.render(x, y, layer);
}

bool Button::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mouseClicked && pressCalldown == 0)
    {
        const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);

        Vec2f mousePos = ne.pos;
        if(m_State == State::Hover)
        {
            m_State = State::Press;
            buttonPressFunc();
            pressCalldown = 10;

            return true;
        }
    }

    return MenuObject::eventCallback(e);
}
```

../src/gui/objects/Button.cpp

gui/objects/Image.h

```
#pragma once

#include "MenuObject.h"

#include "Sprite.h"

class Image : public MenuObject
{
  private:
    Sprite::ID m_SpriteID;

  public:
    Image(float x, float y, float width, float height, Sprite::ID spriteID, Layer *layer);
    Image(std::function<void(float *, float *, float *, float *)> posFunc, Sprite::ID spriteID, Layer *layer);

    virtual void render() override;
    virtual void update() override;
};
```

../src/gui/objects/Image.h

gui/objects/Image.cpp

```
#include "Image.h"

#include "Renderer.h"

Image::Image(float x, float y, float width, float height, Sprite::ID spriteID, Layer *layer)
    : MenuObject(x, y, width, height, layer), m_SpriteID(spriteID)
{
}

Image::Image(std::function<void(float *, float *, float *, float *)> posFunc, Sprite::ID spriteID, Layer *layer)
    : MenuObject(posFunc, layer), m_SpriteID(spriteID)
{
}

void Image::render()
{
    Render::sprite(x, y, 0.0f, width, height, m_SpriteID, 9, true);
}

void Image::update()
{
}
```

../src/gui/objects/Image.cpp

gui/objects/MenuBackgroundObject.h

```
#pragma once

#include "MenuObject.h"

#include <GLM.h>

class MenuBackground : public MenuObject
{
  private:
    glm::vec4             m_Colour;
    std::function<void()> m_ExitFunc;

  public:
    MenuBackground(float x, float y, float width, float height, Layer *layer, glm::vec4 colour, std::function<void()> exitFunc);
    MenuBackground(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, glm::vec4 colour, std::function<void()> exitFunc);
    virtual ~MenuBackground() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
};
```

../src/gui/objects/MenuBackgroundObject.h

gui/objects/MenuBackgroundObject.cpp

```
#include "MenuBackgroundObject.h"

#include "Renderer.h"

MenuBackground::MenuBackground(float x, float y, float width, float height, Layer *layer, glm::vec4 colour, std::function<void()> exitFunc)
    : MenuObject(x, y, width, height, layer), m_Colour(colour), m_ExitFunc(exitFunc)
{
}

MenuBackground::MenuBackground(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, glm::vec4 colour, std::function<void()> exitFunc)
    : MenuObject(posFunc, layer), m_Colour(colour), m_ExitFunc(exitFunc)
{
}

MenuBackground::~MenuBackground()
{
}

void MenuBackground::render()
{
    uint8_t layer = 7;
    Render::rectangle(x, y, 0.0f, width, height, m_Colour, layer, true, true);
}

void MenuBackground::update()
{
}

bool MenuBackground::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::keyInput)
    {
        const Event::KeyboardEvent &ne = static_cast<const Event::KeyboardEvent &>(e);
        if(ne.key == GLFW_KEY_ESCAPE && ne.action == GLFW_PRESS)
        {
            m_ExitFunc();
            return true;
        }
    }
    else if(e.getType() == Event::EventType::mouseClicked)
    {
        const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);
        if(ne.button == GLFW_MOUSE_BUTTON_LEFT && (ne.pos.x < x - width / 2 || ne.pos.x > x + width / 2 || ne.pos.y < y - height / 2 || ne.pos.y > y + height / 2))
        {
            m_ExitFunc();
            return true;
        }
    }
    return MenuObject::eventCallback(e);
}
```

../src/gui/objects/MenuBackgroundObject.cpp

gui/objects/MenuItemHolderManager.h

```
#pragma once

#include "MenuObject.h"

#include <GLM.h>
#include <functional>

#include "Button.h"
#include "Container.h"
#include "Item.h"
#include "Level.h"
#include "TransferObject.h"
#include "Utils.h"

class MIHManager : public MenuObject
{
  private:
    uint32_t                    m_BlockSize;
    IContainer *                         m_Items;
    int *                              m_ActiveItem;
    std::function<void(int, Level *)> m_ClickedFunc;
    glm::vec4                           m_BackgroundColour, m_BorderColour, m_HoverBorderColour, m_ActiveBorderColour;
    Button::State               m_State;

    GUIInventoryIDCode m_ListenID;

  public:   // TODO: Make the function be able to handle it without the level input (cos you can just give it a reference straight to the player)
    MIHManager(float x, float y, float width, float height, float blockSize, Layer *layer, IContainer *items, std::function<void(int, Level *)> clickedFunc, int *activeItem = nullptr, GUIInventoryIDCode listenID = GUIInventoryIDCode::none);
    MIHManager(std::function<void(float *, float *, float *, float *)> posFunc, float blockSize, Layer *layer, IContainer *items, std::function<void(int, Level *)> clickedFunc, int *activeItem = nullptr, GUIInventoryIDCode listenID = GUIInventoryIDCode::none);
    MIHManager(float x, float y, float width, float height, float blockSize, Layer *layer, IContainer *items, glm::vec4 backgroundColour, glm::vec4 borderColour, glm::vec4 hoverColour, glm::vec4 activeColour, std::function<void(int, Level *)> clickedFunc, int *activeItem = nullptr, GUIInventoryIDCode listenID = GUIInventoryIDCode::none);
    MIHManager(std::function<void(float *, float *, float *, float *)> posFunc, float blockSize, Layer *layer, IContainer *items, glm::vec4 backgroundColour, glm::vec4 borderColour, glm::vec4 hoverColour, glm::vec4 activeColour, std::function<void(int, Level *)> clickedFunc, int *activeItem = nullptr, GUIInventoryIDCode listenID = GUIInventoryIDCode::none);
    virtual ~MIHManager() override;

    virtual void render() override;
    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;

    int getIndexMouseAt();

    void transferItem(TransferObject *o);

    void setInventory(IContainer *inventory) { m_Items = inventory; }
};
```

../src/gui/objects/MenuItemHolderManager.h

gui/objects/MenuItemHolderManager.cpp

```
#include "MenuItemHolderManager.h"

#include "Application.h"
#include "Event.h"
#include "GUILayer.h"
#include "Log.h"
#include "Renderer.h"
```

```cpp
#include "ItemContainer.h"
#include "WeaponContainer.h"

#include "MessageManager.h"

MIHManager::MIHManager(float x, float y, float width, float height, float blockSize, Layer *layer, IContainer *items, std::function<void(int, Level *)> clickedFunc, int *activeItem, GUIInventoryIDCode listenID)
    : MenuObject(x, y, width, height, layer),
      m_BlockSize(blockSize),
      m_Items(items),
      m_ActiveItem(activeItem),
      m_ClickedFunc(clickedFunc),
      m_BackgroundColour({0.6f, 0.6f, 0.6f, 1.0f}),
      m_BorderColour({0.0f, 0.0f, 0.0f, 1.0f}),
      m_HoverBorderColour({0.0f, 1.0f, 0.0f, 1.0f}),
      m_ActiveBorderColour({1.0f, 0.0f, 0.0f, 1.0f}),
      m_ListenID(listenID)
{
}
MIHManager::MIHManager(std::function<void(float *, float *, float *, float *)> posFunc, float blockSize, Layer *layer, IContainer *items, std::function<void(int, Level *)> clickedFunc, int *activeItem, GUIInventoryIDCode listenID)
    : MenuObject(posFunc, layer),
      m_BlockSize(blockSize),
      m_Items(items),
      m_ActiveItem(activeItem),
      m_ClickedFunc(clickedFunc),
      m_BackgroundColour({0.6f, 0.6f, 0.6f, 1.0f}),
      m_BorderColour({0.0f, 0.0f, 0.0f, 1.0f}),
      m_HoverBorderColour({0.0f, 1.0f, 0.0f, 1.0f}),
      m_ActiveBorderColour({1.0f, 0.0f, 0.0f, 1.0f}),
      m_ListenID(listenID)

{

}
// TODO: Clean up the parameter order
MIHManager::MIHManager(float x, float y, float width, float height, float blockSize, Layer *layer, IContainer *items, glm::vec4 backgroundColour, glm::vec4 borderColour, glm::vec4 hoverColour, glm::vec4 activeColour, std::function<void(int, Level *)> clickedFunc, int *activeItem, GUIInventoryIDCode listenID)
    : MenuObject(x, y, width, height, layer),
      m_BlockSize(blockSize),
      m_Items(items),
      m_ActiveItem(activeItem),
      m_ClickedFunc(clickedFunc),
      m_BackgroundColour(backgroundColour),
      m_BorderColour(borderColour),
      m_HoverBorderColour(hoverColour),
      m_ActiveBorderColour(activeColour),
      m_ListenID(listenID)

{
}
MIHManager::MIHManager(std::function<void(float *, float *, float *, float *)> posFunc, float blockSize, Layer *layer, IContainer *items, glm::vec4 backgroundColour, glm::vec4 borderColour, glm::vec4 hoverColour, glm::vec4 activeColour, std::function<void(int, Level *)> clickedFunc, int *activeItem, GUIInventoryIDCode listenID)
    : MenuObject(posFunc, layer),
      m_BlockSize(blockSize),
      m_Items(items),
      m_ActiveItem(activeItem),
      m_ClickedFunc(clickedFunc),
      m_BackgroundColour(backgroundColour),
      m_BorderColour(borderColour),
      m_HoverBorderColour(hoverColour),
      m_ActiveBorderColour(activeColour),
      m_ListenID(listenID)
{
}


MIHManager::~MIHManager()
{
    // NOTE: m_Items is a reference to something stored elsewhere, same with m_ActiveItem so they shouldn't be deleted here
}

void MIHManager::render()
{
    if(m_Items)
    {
        int xOffset    = 0;
        int yOffset    = 0;
        int gridWidth  = (int) width / m_BlockSize;
        int gridHeight = (int) height / m_BlockSize;

        uint8_t layer = 7;

        int mouseHoverBlock = -1;
        if(m_State == Button::State::Hover)
        {
            mouseHoverBlock = getIndexMouseAt();
        }
        for(int posY = 0; posY < gridHeight; posY++)
        {
            for(int posX = 0; posX < gridWidth; posX++)
            {
                int   i     = posX + posY * gridWidth;
                float nextX = m_BlockSize / 2 + x + xOffset * m_BlockSize;
                float nextY = m_BlockSize / 2 + y - yOffset * m_BlockSize;

                float borderWidth = 2.0f;
                if(i < m_Items->size() && i == mouseHoverBlock)
                {
                    borderWidth += 1.0f;
                    Render::rectangle(nextX, nextY, m_BlockSize, m_BlockSize, m_BackgroundColour, borderWidth,
    m_HoverBorderColour, layer, true, true);

                    float        scale    = 35.0f;   // TODO: Increase this probably - with scale
                    Vec2f        mousePos = Event::getMousePos();
                    Render::hoverText(*m_Items->getItem(i)->getName(), mousePos.x, mousePos.y, scale, {1.0f, 1.0f,
     1.0f, 1.0f}, {0.3f, 0.3f, 0.3f, 0.7f}, layer + 2, true);
                }
                else if(m_ActiveItem && i == *m_ActiveItem)
                {
                    borderWidth += 3.0f;
                    Render::rectangle(nextX, nextY, m_BlockSize, m_BlockSize, m_BackgroundColour, borderWidth,
    m_ActiveBorderColour, layer, true, true);
                }
                else
                    Render::rectangle(nextX, nextY, m_BlockSize, m_BlockSize, m_BackgroundColour, borderWidth,
    m_BorderColour, layer, true, true);

                if(i < m_Items->size())
                    m_Items->getItem(i)->render(nextX, nextY, 0.0f, m_BlockSize - 10.0f, layer + 1, true);

                xOffset++;
                if(xOffset == gridWidth)
                {
                    xOffset = 0;
                    yOffset++;
                }
                if(yOffset == height)
                    break;
            }
        }
    }
}

void MIHManager::update()
{
    if(m_ActiveItem && (*m_ActiveItem) >= m_Items->size())
        (*m_ActiveItem)--;

    Vec2f mousePos = Event::getMousePos();

    if(mousePos.x > x && mousePos.x < x + width && mousePos.y < y + m_BlockSize && mousePos.y > y + m_BlockSize -
    height)
        m_State = Button::State::Hover;
    else
        m_State = Button::State::None;
}

bool MIHManager::eventCallback(const Event::Event &e)
{
    if(m_Items)
    {
        if(e.getType() == Event::EventType::mouseClicked)
        {
            const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);
            Vec2f mousePos = ne.pos;
            int   hoverBlock = getIndexMouseAt();
            if(ne.button == Event::leftButton && hoverBlock != -1)
            {
                m_State         = Button::State::Press;
                GUILayer *layer = dynamic_cast<GUILayer *>(m_Layer);
                if(layer && hoverBlock < m_Items->size())
                    m_ClickedFunc(hoverBlock, layer->getConnectedLevel());
                return true;
            }
            else if(ne.button == Event::rightButton && hoverBlock != -1)
            {
                Event::ItemTransfer e(hoverBlock, m_Items);
                Application::callEvent(e, true);

                return true;
            }
        }
    }

    if(e.getType() == Event::EventType::chestOpened)
    {
        const Event::ChestOpenedEvent &ne = static_cast<const Event::ChestOpenedEvent &>(e);

        if(m_ListenID == ne.id)
        {
            m_Items = ne.container;
            m_ActiveItem = ne.activeItem;
            return true;
        }

        return false;
    }

    return MenuObject::eventCallback(e);
}

int MIHManager::getIndexMouseAt()
{
    Vec2f mousePos = Event::getMousePos();
    if(mousePos.x > x && mousePos.x < x + width && mousePos.y < y + m_BlockSize && mousePos.y > y + m_BlockSize -
    height)
    {
        int   mouseGridX    = -1;
        int   mouseGridY    = -1;
        Vec2f mousePos      = Event::getMousePos();
        mouseGridX          = (mousePos.x - x) / m_BlockSize;
        mouseGridY          = -(mousePos.y - (y + m_BlockSize)) / m_BlockSize;
        int mouseHoverBlock = mouseGridX + mouseGridY * ((int) width / m_BlockSize);
        if(mouseHoverBlock < 0 || mouseHoverBlock >= ((int) width / m_BlockSize) * ((int) height / m_BlockSize))
            return -1;
        return mouseHoverBlock;
    }
    return -1;
}

void MIHManager::transferItem(TransferObject *o)
{
    int hoverBox = getIndexMouseAt();
    if(hoverBox != -1)
    {
        bool swap = hoverBox < m_Items->size();

        IContainer *oContainer = o->getContainer();
        int         oIndex     = o->getIndex();

        bool cancel = false;

        {
            Weapon *oWeapon = dynamic_cast<Weapon *>(oContainer->getItem(oIndex));
            cancel          = m_Items->getType() == IContainer::Type::weapon && !oWeapon;
        }

        if(swap)
        {
            Weapon *mWeapon = dynamic_cast<Weapon *>(m_Items->getItem(hoverBox));
            cancel          = oContainer->getType() == IContainer::Type::weapon && !mWeapon;
        }

        if(cancel)
        {
            MessageManager::sendMessage("Item cannot be stored there!", MessageManager::Priority::High);
            return;
        }

        if(!(oContainer == m_Items && !swap))
        {
            auto insertItem = [swap](IContainer *container, Item *item, int index) {
                switch(container->getType())
                {
                case IContainer::Type::item:
                {
                    ItemContainer *itemContainer = static_cast<ItemContainer *>(container);

                    if(item == nullptr)
                    {
                        itemContainer->erase(itemContainer->begin() + index);
                        break;
                    }

                    if(swap)
                    {
                        itemContainer->erase(itemContainer->begin() + index);
                        itemContainer->insert(itemContainer->begin() + index, item);
                    }
                    else
                        itemContainer->push_back(item);
                    break;
                }
                case IContainer::Type::weapon:
                {
                    WeaponContainer *weaponContainer = static_cast<WeaponContainer *>(container);

                    if(item == nullptr)
                    {
                        weaponContainer->erase(weaponContainer->begin() + index);
                        break;
                    }

                    Weapon *weapon = static_cast<Weapon *>(item);
                    if(swap)
                    {
                        weaponContainer->erase(weaponContainer->begin() + index);
                        weaponContainer->insert(weaponContainer->begin() + index, weapon);
                    }
                    else
                        weaponContainer->push_back(weapon);
                    break;
                }
                default:
                    Log::warning("Unknown container type!");
                    break;
                }
            };

            Item *oItem = oContainer->getItem(oIndex);
            Item *mItem = nullptr;
            if(swap)
                mItem = m_Items->getItem(hoverBox);

            insertItem(m_Items, oItem, hoverBox);
            insertItem(oContainer, mItem, oIndex);

            if(m_ActiveItem && (*m_ActiveItem) == -1 && m_Items->size() > 0)
                (*m_ActiveItem) = 0;
        }
        o->hasTransferred();
    }
}
```

../src/gui/objects/MenuItemHolderManager.cpp

gui/objects/StatBar.h

```
#pragma once

#include "MenuObject.h"

class StatBar : public MenuObject
{   // TODO: Add name and will appear when mouse is over it
  private:
    const float *m_Stat, *m_StatMax;
    glm::vec4     m_BackgroundColour;

  public:
    StatBar(float x, float y, float width, float height, Layer *layer, const float *stat, const float *statMax);
    StatBar(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, const float *stat,
    const float *statMax);
    StatBar(float x, float y, float width, float height, Layer *layer, const float *stat, const float *statMax,
    glm::vec4 backgroundColour);
    StatBar(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, const float *stat,
    const float *statMax, glm::vec4 backgroundColour);
    virtual ~StatBar() override;

    virtual void update() override;
    virtual void render() override;
    virtual bool eventCallback(const Event::Event &e) override;

    void setStat(float *stat, float *statMax);
};
```

../src/gui/objects/StatBar.h

gui/objects/StatBar.cpp

```
#include "StatBar.h"

#include "Renderer.h"

StatBar::StatBar(float x, float y, float width, float height, Layer *layer, const float *stat, const float *
    statMax)
    : MenuObject(x, y, width, height, layer), m_Stat(stat), m_StatMax(statMax), m_BackgroundColour({1.0f, 0.0f,
    0.0f, 1.0f})
{
}

StatBar::StatBar(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, const float *stat,
     const float *statMax)
    : MenuObject(posFunc, layer), m_Stat(stat), m_StatMax(statMax), m_BackgroundColour({1.0f, 0.0f, 0.0f, 1.0f})
{
}

StatBar::StatBar(float x, float y, float width, float height, Layer *layer, const float *stat, const float *
    statMax, glm::vec4 backgroundColour)
    : MenuObject(x, y, width, height, layer), m_Stat(stat), m_StatMax(statMax), m_BackgroundColour(
    backgroundColour)
{
}

StatBar::StatBar(std::function<void(float *, float *, float *, float *)> posFunc, Layer *layer, const float *stat,
     const float *statMax, glm::vec4 backgroundColour)
    : MenuObject(posFunc, layer), m_Stat(stat), m_StatMax(statMax), m_BackgroundColour(backgroundColour)
{
}

StatBar::~StatBar()
{
    // NOTE: m_Stat and m_StatMax should be references to variables stored elsewhere, thus they should not be
    deleted here!
}

void StatBar::update()
{
}

void StatBar::render()
{
    uint8_t layer = 7;
    Render::rectangle(x, y, 0.0f, width * (*m_Stat) / (*m_StatMax), height, m_BackgroundColour, layer, true, true)
    ;
}

bool StatBar::eventCallback(const Event::Event &e)
{
    return MenuObject::eventCallback(e);
}

void StatBar::setStat(float *stat, float *statMax)
{
    m_Stat    = stat;
    m_StatMax = statMax;
}
```

../src/gui/objects/StatBar.cpp

gui/objects/Text.h

```
#pragma once

#include <GLM.h>
#include <string>

struct Text
{
    float x, y;

    std::string m_Text;
    float       m_Scale;
    glm::vec4   m_Colour;

    bool isCentered, isOverlay;

    Text(std::string text);
    Text(std::string text, float x, float y, float scale, glm::vec4 colour, bool isCentered = true, bool isOverlay
     = true);
    ~Text();

    void render(float xOffset, float yOffset, uint8_t layer);
};
```

../src/gui/objects/Text.h

gui/objects/Text.cpp

```
#include "Text.h"

#include "Renderer.h"

Text::Text(std::string text)
    : x(0.0f), y(0.0f), m_Text(text), m_Scale(50.0f), m_Colour({1.0f, 1.0f, 1.0f, 1.0f}), isCentered(true),
    isOverlay(true)
{
}

Text::Text(std::string text, float x, float y, float scale, glm::vec4 colour, bool isCentered, bool isOverlay)
    : x(x), y(y), m_Text(text), m_Scale(scale), m_Colour(colour), isCentered(isCentered), isOverlay(isOverlay)
{
}

Text::~Text()
{
}

void Text::render(float xOffset, float yOffset, uint8_t layer)
{
    Render::text(m_Text, x + xOffset, y + yOffset, m_Scale, m_Colour, layer, isCentered, isOverlay);
}
```

../src/gui/objects/Text.cpp

gui/objects/TextMenuObject.h

```
#pragma once

#include "MenuObject.h"
#include "Text.h"

class TextMenuObject : public MenuObject
{
  private:
    Text m_Text;

  public:
    TextMenuObject(Text text, float x, float y, Layer *layer);
    TextMenuObject(Text text, std::function<void(float *, float *)> posFunc, Layer *layer);

    virtual void render() override;
    virtual void update() override;
};
```

../src/gui/objects/TextMenuObject.h

gui/objects/TextMenuObject.cpp

```cpp
#include "TextMenuObject.h"

TextMenuObject::TextMenuObject(Text text, float x, float y, Layer *layer)
    : MenuObject(x, y, 0.0f, 0.0f, layer), m_Text(text)
{
}

TextMenuObject::TextMenuObject(Text text, std::function<void(float *, float *)> posFunc, Layer *layer)
    : MenuObject([posFunc](float *x, float *y, float *width, float *height) { posFunc(x, y); (*width) = 0.0f; (*height) = 0.0f; }, layer), m_Text(text)
{
}

void TextMenuObject::render()
{
    m_Text.render(x, y, 8);
}

void TextMenuObject::update()
{
}
```

../src/gui/objects/TextMenuObject.cpp

gui/objects/TransferObject.h

```cpp
#pragma once

#include "Container.h"
#include "Item.h"
#include "MenuObject.h"

class TransferObject : public MenuObject
{
  private:
    uint16_t    index;
    IContainer *container;

  public:
    TransferObject(float width, float height, Layer *layer);
    virtual ~TransferObject();

    virtual void update() override;
    virtual void render() override;
    virtual bool eventCallback(const Event::Event &e) override;

    uint16_t    getIndex() { return index; }
    IContainer *getContainer() { return container; }

    void hasTransferred();
};
```

../src/gui/objects/TransferObject.h

gui/objects/TransferObject.cpp

```cpp
#include "TransferObject.h"

#include "Event.h"
#include "GUILayer.h"

#include <vector>

TransferObject::TransferObject(float width, float height, Layer *layer)
    : MenuObject(0.0f, 0.0f, width, height, layer), index(0), container(nullptr)
{
}

TransferObject::~TransferObject()
{
}

void TransferObject::update()
{
    if(container)
    {
        Vec2f mousePos = Event::getMousePos();

        x = mousePos.x;
        y = mousePos.y;
    }
}
```

```cpp
void TransferObject::render()
{
    if(container)
    {
        container->getItem(index)->render(x, y, 0.0f, width, 9, true);
    }
}

bool TransferObject::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::itemTransfer)
    {
        const Event::ItemTransfer &ne = static_cast<const Event::ItemTransfer &>(e);

        index     = ne.index;
        container = ne.container;
        // TODO: Make it so it doesn't render item in other places

        return true;
    }
    if(container)
    {
        if(e.getType() == Event::EventType::mouseClicked)
        {
            GUILayer *layer = dynamic_cast<GUILayer *>(m_Layer);
            if(layer)
            {
                layer->transferObject(this);
            }
            else
                Log::warning("Transfer object on incorrect layer!");
            return true;
        }
        else if(e.getType() == Event::EventType::keyInput)
        {
            const Event::KeyboardEvent &ne = static_cast<const Event::KeyboardEvent &>(e);

            if(ne.key == GLFW_KEY_ESCAPE && (ne.action == GLFW_PRESS || ne.action == GLFW_REPEAT))
            {
                hasTransferred();
                return true;
            }
        }
    }
    return MenuObject::eventCallback(e);
}

void TransferObject::hasTransferred()
{
    container = nullptr;
    index     = 0;
}
```

../src/gui/objects/TransferObject.cpp

3.11 Entities

3.11.1 Base

entity/Entity.h

```cpp
#pragma once

#include "Event.h"
#include "Log.h"
#include "Sprite.h"
#include "Utils.h"

class Level;
class Entity
{
  protected:
    float        x, y;
    float        width, height;
    Sprite::ID   m_SpriteID;
    Level *      m_Level;
    CollisionBox m_CollisionBox;
  public:
    Entity();
    Entity(float x, float y, float size, Sprite::ID spriteID);
    Entity(float x, float y, float size, Level *level, Sprite::ID spriteID);
    Entity(float x, float y, float size, CollisionBox box, Level *level, Sprite::ID spriteID);
    virtual ~Entity();

    virtual void update()                             = 0;
    virtual void render()                             = 0;
    virtual bool eventCallback(const Event::Event &e);

    float        getX() const;
    float        getY() const;
    float         getWidth() const;
    float         getHeight() const;
    virtual bool getIsMoving();
    CollisionBox &getCollisionBox() { return m_CollisionBox; }
    bool doesIntersectWith(Vec2f pos);
    virtual bool  hasCollidedWith(float xs, float ys, CollisionBox box);
    virtual bool deleteMe();

    virtual void changeX(float changeBy);
    virtual void changeY(float changeBy);
    void         setLevel(Level *level);


#ifdef DEBUG
    virtual void imGuiRender();
#endif
};
```

../src/entity/Entity.h

entity/Entity.cpp

```
#include "Entity.h"

#include "KeyDefinitions.h"
#include "Level.h"

Entity::Entity()
    : x(0.0f), y(0.0f), width(TILE_SIZE), height(TILE_SIZE), m_Level(nullptr), m_CollisionBox({{0.0f, 0.0f}, {0.0f, 0.0f}}), m_SpriteID(Sprite::ID::errorID) {}
Entity::Entity(float x, float y, float size, Sprite::ID spriteID)
    : x(x), y(y), width(size), height(size), m_Level(nullptr), m_CollisionBox({{-size / 2, -size / 2}, {size / 2, size / 2}}), m_SpriteID(spriteID) {}
Entity::Entity(float x, float y, float size, Level *level, Sprite::ID spriteID)
    : x(x), y(y), width(size), height(size), m_Level(level), m_CollisionBox({{-size / 2, -size / 2}, {size / 2, size / 2}}), m_SpriteID(spriteID) {}
Entity::Entity(float x, float y, float size, CollisionBox box, Level *level, Sprite::ID spriteID)
    : x(x), y(y), width(size), height(size), m_Level(level), m_CollisionBox(box), m_SpriteID(spriteID) {}
Entity::~Entity() {}

bool Entity::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        const Event::MazeMovedEvent &ne = static_cast<const Event::MazeMovedEvent &>(e);
        x += ne.changeX;
        y += ne.changeY;
    }
    return false;
}

float Entity::getX() const { return x; }
float Entity::getY() const { return y; }
float Entity::getWidth() const { return width; }
float Entity::getHeight() const { return height; }
bool  Entity::getIsMoving() { return false; }

bool Entity::doesIntersectWith(Vec2f pos)
{
    return doesPointIntersectWithBox(pos, {x, y}, m_CollisionBox);
}
bool Entity::deleteMe() { return false; }

void Entity::changeX(float changeBy) { x += changeBy; }
void Entity::changeY(float changeBy) { y += changeBy; }
void Entity::setLevel(Level *level) { m_Level = level; }

bool Entity::hasCollidedWith(float xs, float ys, CollisionBox box)
```

```
{
    if(xs + box.lowerBound.x >= x + m_CollisionBox.upperBound.x || x + m_CollisionBox.lowerBound.x >= xs + box.
    upperBound.x)
        return false;
    if(ys + box.upperBound.y <= y + m_CollisionBox.lowerBound.y || y + m_CollisionBox.upperBound.y <= ys + box.
    lowerBound.y)
        return false;
    return true;
}

#ifdef DEBUG
void Entity::imGuiRender()
{
}
#endif
```

../src/entity/Entity.cpp

entity/movableEntity/MovableEntity.h

```
#pragma once

#include "Entity.h"

class MovableEntity : public Entity
{
  protected:
    float     m_Speed;
    bool      isMoving;
    Direction m_Dir;

    bool isGhost;

  public:
    MovableEntity();
    MovableEntity(float x, float y, float size, Level *level, Sprite::ID spriteID);
    MovableEntity(float x, float y, float size, CollisionBox box, Level *level, Sprite::ID spriteID);
    MovableEntity(float x, float y, float size, float speed, Direction dir, CollisionBox box, Level *level, Sprite
    ::ID spriteID);
    virtual ~MovableEntity() override;

    virtual bool eventCallback(const Event::Event &e) override;

    virtual void move(float xa, float ya);
    virtual void move(Vec2f ratio);

    bool canMove(float xa, float ya);
    bool canMove(Vec2f ratio);

    virtual bool      getIsMoving() override { return isMoving; }
    virtual Direction getDirection() { return m_Dir; }
    CollisionBox      getMovingCollisionBox();
};
```

../src/entity/movableEntity/MovableEntity.h

entity/movableEntity/MovableEntity.cpp

```
#include "MovableEntity.h"

#include "Level.h"

MovableEntity::MovableEntity()
    : Entity(), m_Speed(7.0f), isMoving(false), m_Dir(Direction::south), isGhost(false) {}

MovableEntity::MovableEntity(float x, float y, float size, Level *level, Sprite::ID spriteID)
    : Entity(x, y, size, level, spriteID), m_Speed(7.0f), isMoving(false), m_Dir(Direction::south), isGhost(false)
     {}

MovableEntity::MovableEntity(float x, float y, float size, CollisionBox box, Level *level, Sprite::ID spriteID)
    : Entity(x, y, size, box, level, spriteID), m_Speed(7.0f), isMoving(false), m_Dir(Direction::south), isGhost(
    false) {}

MovableEntity::MovableEntity(float x, float y, float size, float speed, Direction dir, CollisionBox box, Level *
    level, Sprite::ID spriteID)
    : Entity(x, y, size, box, level, spriteID), m_Speed(speed), isMoving(false), m_Dir(dir), isGhost(false) {}

MovableEntity::~MovableEntity() {}

void MovableEntity::move(float xa, float ya)
{
    if(!isGhost)
    {
        auto [colX, colY] = m_Level->directionalCollisionCheck(x, y, xa, ya, getMovingCollisionBox());
        if(colX)
            xa = 0;
        if(colY)
            ya = 0;
        if(xa == 0 && ya == 0)
        {
            isMoving = false;
            return;
        }
    }
    isMoving = true;
    x += xa;
    y += ya;
    if(fabs(xa) > fabs(ya))
    {
        if(ya < 0)
            m_Dir = Direction::south;
        else if(ya > 0)
            m_Dir = Direction::north;
        if(xa < 0)
            m_Dir = Direction::west;
        else if(xa > 0)
            m_Dir = Direction::east;
    }
    else
    {
        if(xa < 0)
            m_Dir = Direction::west;
        else if(xa > 0)
            m_Dir = Direction::east;
        if(ya < 0)
            m_Dir = Direction::south;
        else if(ya > 0)
            m_Dir = Direction::north;
    }
}

void MovableEntity::move(Vec2f ratio)
{
    if(ratio.y == 0)
        move(ratio.x > 0 ? m_Speed : -m_Speed, 0.0f);
    else if(ratio.x == 0)
        move(0.0f, ratio.y > 0 ? m_Speed : -m_Speed);
    else
    {
        float speedSquared = m_Speed * m_Speed;
        float sum          = std::fabs(ratio.x) + std::fabs(ratio.y);
        float sumSquared   = sum * sum;
        float timesBy      = std::sqrt((sumSquared * speedSquared) / (ratio.x * ratio.x + ratio.y * ratio.y));
        move((ratio.x * timesBy) / sum, (ratio.y * timesBy) / sum);
    }
}

bool MovableEntity::canMove(float xa, float ya)
{

    auto [colX, colY] = m_Level->directionalCollisionCheck(x, y, xa, ya, getMovingCollisionBox());
    if(colX)
        xa = 0;
    if(colY)
        ya = 0;
    return !(xa == 0 && ya == 0);
}

bool MovableEntity::canMove(Vec2f ratio)
{
    if(ratio.y == 0)
        return canMove(m_Speed, 0.0f);
    else if(ratio.x == 0)
        return canMove(0.0f, m_Speed);
    else
    {
        float speedSquared = m_Speed * m_Speed;
        float sum          = std::fabs(ratio.x) + std::fabs(ratio.y);
        float sumSquared   = sum * sum;
        float timesBy      = std::sqrt((sumSquared * speedSquared) / (ratio.x * ratio.x + ratio.y * ratio.y));
        return canMove((ratio.x * timesBy) / sum, (ratio.y * timesBy) / sum);
    }
```

```
}

bool MovableEntity::eventCallback(const Event::Event &e)
{
    return Entity::eventCallback(e);
}

CollisionBox MovableEntity::getMovingCollisionBox()
{
    return {m_CollisionBox.lowerBound,
            {m_CollisionBox.upperBound.x, m_CollisionBox.upperBound.y / 4.0f}};
}
```

../src/entity/movableEntity/MovableEntity.cpp

3.11.2 Mob

entity/movableEntity/mob/Mob.h

```
#pragma once

#include "MovableEntity.h"

#include "AnimatedSprite.h"
#include "Container.h"
#include "Item.h"
#include "ItemContainer.h"
#include "StatsMob.h"
#include "Utils.h"
#include "Weapon.h"
#include "WeaponContainer.h"

class Mob : public MovableEntity, public StatsMob
{
  protected:
    ItemContainer          m_Inventory;
    WeaponContainer        m_Weapons;
    int                    m_CurrentWeapon;

    Container<Mob *> m_Followers;
    Mob *            m_Following;
    Mob *            m_Enemy;

    std::unique_ptr<AnimatedSprite> m_NorthAnimation;
    std::unique_ptr<AnimatedSprite> m_SouthAnimation;
    std::unique_ptr<AnimatedSprite> m_EastAnimation;
    std::unique_ptr<AnimatedSprite> m_WestAnimation;

    bool isInControl = false;

    void setupAnimations();

  public:
    Mob();
    Mob(float x, float y);
    Mob(float x, float y, Level *level);
    Mob(float x, float y, Level *level, Sprite::ID spriteID);
    Mob(float x, float y, float speed, Level *level, Sprite::ID spriteID);
    virtual ~Mob() override;

    bool pickUp(Item *item);

    virtual void render();
    virtual void update();
    virtual bool eventCallback(const Event::Event &e);

    virtual bool deleteMe() { return isDead(); }

    ItemContainer *  getInventory() { return &m_Inventory; }
    WeaponContainer *getWeapons() { return &m_Weapons; }
    int *            getCurrentWeaponPointer() { return &m_CurrentWeapon; }
    void             setCurrentWeapon(int currentWeapon) { m_CurrentWeapon = currentWeapon; }
    void             useItemInInventory(uint16_t index);

    bool addFollower(Mob *follower);
    void removeFollower(Mob *follower);
    bool canAddFollower();

    virtual void setFollowing(Mob *following);
    virtual void setEnemy(Mob *enemy);
    void setFollowersEnemy(Mob *enemy);

    const Container<Mob *> &getFollowers() const { return m_Followers; }
    const Mob *             getFollowing() const { return m_Following; }
    const Mob *             getEnemy() const { return m_Enemy; }

    void useCurrentWeapon(bool hold);

    void setIsInControl(bool i_isInControl)
    {
        isInControl = i_isInControl;
    }
#ifdef DEBUG
    virtual void imGuiRender() = 0;
#endif
};
```

../src/entity/movableEntity/mob/Mob.h

entity/movableEntity/mob/Mob.cpp

```
#include "Mob.h"

#include "Potion.h"
#include "Weapon.h"

#include "Application.h"
#include "KeyDefinitions.h"
#include "Level.h"
#include "Renderer.h"

#include <math.h>

#define defaultBox                              \
    {                                           \
        {                                       \
            (float) TILE_SIZE * 0.37f,   \
            (float) -TILE_SIZE * 0.04f}, \
        {                                       \
            (float) TILE_SIZE * 0.6f,    \
                (float) TILE_SIZE * 1.0f \
        }                                       \
    }

#define DEFINE_MY_VARS m_Inventory(DEFAULT_INVENTORY_SIZE), m_Weapons(3), m_CurrentWeapon(-1), m_Followers(1),
    m_Following(nullptr), m_Enemy(nullptr)

Mob::Mob()
    : MovableEntity(0.0f, 0.0f, TILE_SIZE * 1.25f, defaultBox, nullptr, Sprite::ID::mobPlayer), StatsMob(),
    DEFINE_MY_VARS
{
    setupAnimations();
}

Mob::Mob(float x, float y)
    : MovableEntity(x, y, TILE_SIZE * 1.25f, defaultBox, nullptr, Sprite::ID::mobPlayer), StatsMob(),
    DEFINE_MY_VARS
{
    setupAnimations();
}

Mob::Mob(float x, float y, Level *level)
    : MovableEntity(x, y, TILE_SIZE * 1.25f, defaultBox, level, Sprite::ID::mobPlayer), StatsMob(), DEFINE_MY_VARS
{
    setupAnimations();
}

Mob::Mob(float x, float y, Level *level, Sprite::ID spriteID)
    : MovableEntity(x, y, TILE_SIZE * 1.25f, defaultBox, level, spriteID), StatsMob(), DEFINE_MY_VARS
{
    setupAnimations();
}

Mob::Mob(float x, float y, float speed, Level *level, Sprite::ID spriteID)
    : MovableEntity(x, y, TILE_SIZE * 1.25f, speed, Direction::south, defaultBox, level, spriteID), StatsMob(),
    DEFINE_MY_VARS
{
    setupAnimations();
}
```

```cpp
Mob::~Mob()
{
}

void Mob::render()
{
    uint8_t layer = 4;
    if(isMoving)
    {
        switch(m_Dir)
        {
        case Direction::north:
            m_NorthAnimation->render(x, y, 0.0f, width, height, layer);
            break;
        case Direction::south:
            m_SouthAnimation->render(x, y, 0.0f, width, height, layer);
            break;
        case Direction::east:
            m_EastAnimation->render(x, y, 0.0f, width, height, layer);
            break;
        default:
            m_WestAnimation->render(x, y, 0.0f, width, height, layer);
            break;
        }
    }
    else
    {
        switch(m_Dir)
        {
        case Direction::north:
            Render::sprite(x, y, 0.0f, width, height, m_SpriteID + SPRITE_NORTH, layer);
            break;
        case Direction::south:
            Render::sprite(x, y, 0.0f, width, height, m_SpriteID + SPRITE_SOUTH, layer);
            break;
        case Direction::east:
            Render::sprite(x, y, 0.0f, width, height, m_SpriteID + SPRITE_EAST, layer);
            break;
        default:
            Render::sprite(x, y, 0.0f, width, height, m_SpriteID + SPRITE_WEST, layer);
            break;
        }
    }
}

void Mob::update()
{
    if(!isDead())
    {
        if(isMoving)
        {
            switch(m_Dir)
            {
            case Direction::north:
                m_NorthAnimation->update();
                break;
            case Direction::south:
                m_SouthAnimation->update();
                break;
            case Direction::east:
                m_EastAnimation->update();
                break;
            default:
                m_WestAnimation->update();
                break;
            }
        }

        for(Weapon *w : m_Weapons)
            static_cast<Weapon *>(w)->update();

        StatsMob::update();
    }
    else
    {
        Event::MobDied *e = new Event::MobDied(this);
        Application::callEventLater(e);
    }
}
```

```cpp
bool Mob::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mobDied)
    {
        const Event::MobDied &ne = static_cast<const Event::MobDied &>(e);

        if(ne.mob == m_Following)
            m_Following = nullptr;
        else if(ne.mob == m_Enemy)
            m_Enemy = nullptr;
        else
        {
            auto index = std::find(m_Followers.begin(), m_Followers.end(), ne.mob);
            if(index != m_Followers.end())
                m_Followers.erase(index);
        }

        if(m_Enemy)
        {
            setFollowersEnemy(m_Enemy);
            m_Level->getPlayer()->setFollowersEnemy(this);
        }
    }
    else if(!m_Following && e.getType() == Event::EventType::showAltTileEvent)
    {
        const Event::ShowAltTileEvent &ne = static_cast<const Event::ShowAltTileEvent &>(e);
        if(ne.showAlt)
        {
            if(m_Enemy)
            {
                setFollowersEnemy(m_Enemy);
                m_Level->getPlayer()->setFollowersEnemy(this);
            }
        }
    }
    return MovableEntity::eventCallback(e);
}

void Mob::useItemInInventory(uint16_t index)
{
    if(index > m_Inventory.size())
    {
        Log::warning("Trying to access item outside of the inventory!");
        return;
    }

    Item *  item = m_Inventory[index];
    Weapon *wp   = dynamic_cast<Weapon *>(item);
    if(wp)
    {
        Event::ItemTransfer e(index, &m_Inventory);
        Application::callEvent(e, true);
        return;
    }

    Potion *pt = dynamic_cast<Potion *>(item);
    if(pt)
    {
        pt->useOn(this);
        m_Inventory.erase(m_Inventory.begin() + index);
        return;
    }

    // NOTE: This is where other stuff will go :D
}

bool Mob::addFollower(Mob *follower)
{
    if(m_Followers.isFull())
        return false;

    follower->setFollowing(this);
    m_Followers.push_back(follower);
    return true;
}

void Mob::removeFollower(Mob *follower)
{
    auto index = std::find(m_Followers.begin(), m_Followers.end(), follower);
    if(index != m_Followers.end())
    {
        m_Followers.erase(index);
        follower->setFollowing(nullptr);
    }
    else
        Log::warning("Cannot find follower to remove!");
}

bool Mob::canAddFollower()
{
    return !m_Followers.isFull();
}

void Mob::setFollowing(Mob *following)
{
    m_Following = following;
}

void Mob::setEnemy(Mob *enemy)
{
    m_Enemy = enemy;
}

void Mob::setFollowersEnemy(Mob *enemy)
{
    if(m_Followers.size() > 0)
    {
        const Container<Mob *> &eFollowers = enemy->getFollowers();

        int i = 0;
        for(Mob *follower : m_Followers)
        {
            if(i == eFollowers.size())
            {
                if(!follower->getEnemy())
                    follower->setEnemy(enemy);
                i = -1;
            }
            else if(!follower->getEnemy())
                follower->setEnemy(eFollowers[i]);

            i++;
        }
    }
}

void Mob::useCurrentWeapon(bool hold)
{
    Weapon *weapon = m_Weapons[m_CurrentWeapon];
    weapon->attack(m_Level, *this, m_Dir, hold);
}

void Mob::setupAnimations()
{
    m_NorthAnimation = std::make_unique<AnimatedSprite>(ANIMATION_FRAMES, m_SpriteID + SPRITE_NORTH);
    m_SouthAnimation = std::make_unique<AnimatedSprite>(ANIMATION_FRAMES, m_SpriteID + SPRITE_SOUTH);
    m_EastAnimation  = std::make_unique<AnimatedSprite>(ANIMATION_FRAMES, m_SpriteID + SPRITE_EAST);
    m_WestAnimation  = std::make_unique<AnimatedSprite>(ANIMATION_FRAMES, m_SpriteID + SPRITE_WEST);
}

bool Mob::pickUp(Item *item)
{
    Weapon *weapon = dynamic_cast<Weapon *>(item);
    if(weapon && !m_Weapons.isFull())
    {
        m_Weapons.push_back(weapon);
        if(m_CurrentWeapon == -1)
            m_CurrentWeapon = 0;
        return true;
    }
    else if(!m_Inventory.isFull())
    {
        m_Inventory.push_back(item);
        return true;
    }

    return false;
}
```

../src/entity/movableEntity/mob/Mob.cpp

entity/movableEntity/mob/StatsMob.h

```
#pragma once

class StatsMob
{
  private:
    float m_Stat_Health, m_Stat_MaxHealth;
    float m_Stat_Stamina, m_Stat_MaxStamina;

    uint16_t m_Stat_RegenDelay, m_Stat_MaxRegenDelay;

    int m_Stat_Strength, m_Stat_Agility;

    int      m_Stat_CombatAbility;
    int      m_Stat_Confidence;
    uint16_t m_Stat_Boredom;
    uint16_t m_Stat_Attractiveness;
    uint16_t m_Stat_MaxActiveWeapons;
    uint16_t m_Stat_MaxInventory;

  public:
    StatsMob();
    virtual ~StatsMob();

    virtual void update();

    void  dealDamage(float damage);
    void  changeHealth(float changeBy);
    void  changeMaxHealth(float changeBy);
    void  changeStamina(float changeBy);
    void  changeConfidence(float changeBy);
    void  changeBoredom(float changeBy);
    float getDamage(float minDamage, float maxDamage);
    void  hasHitTarget(float damageDealt);
    void  hasMissedTarget();
    void  hasUsedWeapon();
    void  resetStats();

    int getWeaponDelay(int delay);

    bool     isDead();
    float    getHealth();
    float    getMaxHealth();
    float    getStamina();
    float    getMaxStamina();
    float    getMaxActiveWeapons();
    float    getMaxInventory();
    uint16_t getAttractiveness();

    const float *getHealthPointer();
    const float *getMaxHealthPointer();
    const float *getStaminaPointer();
    const float *getMaxStaminaPointer();
};
```

../src/entity/movableEntity/mob/StatsMob.h

entity/movableEntity/mob/StatsMob.cpp

```
#include "StatsMob.h"

#include "Log.h"
#include "RandomGen.h"
#include "Utils.h"
#include "math.h"

#define MAXLEVEL            15
#define BINOMIAL_PRECISION  10

StatsMob::StatsMob()
    : m_Stat_Health(100),
      m_Stat_MaxHealth(100),
      m_Stat_Stamina(100),
      m_Stat_MaxStamina(100),
      m_Stat_RegenDelay(0),
      m_Stat_MaxRegenDelay(60),
      m_Stat_Strength(5),
      m_Stat_Agility(5),
      m_Stat_CombatAbility(5),
      m_Stat_Boredom(0),
      m_Stat_Confidence(100),
```

```cpp
      m_Stat_Attractiveness(1),
      m_Stat_MaxActiveWeapons(3),
      m_Stat_MaxInventory(20)   // NOTE: This should not be changed
{
}   // TODO: Once killed a mob have int increase experience, which after a while increases stats
// TODO: Trapdoor room spawns enemy with 2 followers
// TODO: Followed sets the distance a follower should keep

StatsMob::~StatsMob()
{
}

void StatsMob::update()
{
    if(m_Stat_RegenDelay > 0)
        m_Stat_RegenDelay--;
    else
    {
        changeHealth(1);
        changeStamina(5);
        m_Stat_RegenDelay = 10;
    }
}

void StatsMob::dealDamage(float damage)
{
    // TODO: Have strength and agility lower this
    if(m_Stat_Health < damage)
        m_Stat_Health = 0;
    else
        m_Stat_Health -= damage;

    m_Stat_RegenDelay = 100;
}

void StatsMob::changeHealth(float changeBy)
{
    if(changeBy < 0 && m_Stat_Health < -changeBy)
        m_Stat_Health = 0;
    else if(m_Stat_Health + changeBy > m_Stat_MaxHealth)
        m_Stat_Health = m_Stat_MaxHealth;
    else
        m_Stat_Health += changeBy;
}

void StatsMob::changeMaxHealth(float changeBy)
{
    m_Stat_MaxHealth += changeBy;
}

void StatsMob::changeStamina(float changeBy)
{
    if(changeBy < 0 && m_Stat_Stamina < -changeBy)
        m_Stat_Stamina = 0;
    else if(m_Stat_Stamina + changeBy > m_Stat_MaxStamina)
        m_Stat_Stamina = m_Stat_MaxStamina;
    else
        m_Stat_Stamina += changeBy;
}

void StatsMob::changeConfidence(float changeBy)
{
    if(m_Stat_Confidence + changeBy < 0)
        m_Stat_Confidence = 0;
    else if(m_Stat_Confidence + changeBy > 100)
        m_Stat_Confidence = 100;
    else
        m_Stat_Confidence += changeBy;
}

void StatsMob::changeBoredom(float changeBy)
{
    if(m_Stat_Boredom + changeBy < 0)
        m_Stat_Boredom = 0;
    else if(m_Stat_Boredom + changeBy > 100)
        m_Stat_Boredom = 100;
    else
        m_Stat_Boredom += changeBy;
}
```

```
float StatsMob::getDamage(float minDamage, float maxDamage)
{
    if(maxDamage < minDamage)
        Log::info("Max damage is bigger then min damage! What are you doing with your life?");

    float  range       = maxDamage - minDamage;
    double probability = ((double) (2 * m_Stat_CombatAbility + m_Stat_Strength) / (3 * MAXLEVEL)) * (((double)
    m_Stat_Confidence + 100.0f) / 200.0f) * 3 / (m_Stat_Boredom + 3);

    std::vector<float> weights;
    for(int i = 0; i < BINOMIAL_PRECISION + 1; i++)
        weights.push_back(factorial(BINOMIAL_PRECISION) / (factorial(i) * factorial(BINOMIAL_PRECISION - i)) * pow
    (probability, i) * pow(1 - probability, BINOMIAL_PRECISION - i));

    uint32_t randomNum  = Random::getWeightedNum(weights);
    float    percentage = (float) randomNum / BINOMIAL_PRECISION;

    return minDamage + range * percentage;
}

void StatsMob::hasHitTarget(float damageDealt)
{
    changeConfidence(1 + damageDealt / 10);
    changeBoredom(-5);
    changeStamina(-10);
}

void StatsMob::hasMissedTarget()
{
    changeConfidence(-2);
    changeStamina(-10);
}

void StatsMob::hasUsedWeapon()
{
    m_Stat_RegenDelay = m_Stat_MaxRegenDelay;
}

int StatsMob::getWeaponDelay(int delay)
{
    float percentage = 1.0f - (2 * ((float) m_Stat_Agility / MAXLEVEL) + ((float) m_Stat_Stamina /
    m_Stat_MaxStamina)) / 4;
    return delay * percentage;
}

bool     StatsMob::isDead() { return m_Stat_Health == 0; }
float    StatsMob::getHealth() { return m_Stat_Health; }
float    StatsMob::getMaxHealth() { return m_Stat_MaxHealth; }
float    StatsMob::getStamina() { return m_Stat_Stamina; }
float    StatsMob::getMaxStamina() { return m_Stat_MaxStamina; }
float    StatsMob::getMaxActiveWeapons() { return m_Stat_MaxActiveWeapons; }
float    StatsMob::getMaxInventory() { return m_Stat_MaxInventory; }
uint16_t StatsMob::getAttractiveness() { return m_Stat_Attractiveness; }

const float *StatsMob::getHealthPointer() { return &m_Stat_Health; }
const float *StatsMob::getMaxHealthPointer() { return &m_Stat_MaxHealth; }
const float *StatsMob::getStaminaPointer() { return &m_Stat_Stamina; }
const float *StatsMob::getMaxStaminaPointer() { return &m_Stat_MaxStamina; }

void StatsMob::resetStats()
{
    m_Stat_Health = m_Stat_MaxHealth;
}
```

../src/entity/movableEntity/mob/StatsMob.cpp

entity/movableEntity/mob/Player.h

```
#pragma once

#include "AnimatedSprite.h"
#include "Mob.h"

class Player : public Mob
{
  public:
    Player();
    Player(float x, float y);
    Player(float x, float y, Level *level);
    virtual ~Player() override;
    virtual void update() override;
    virtual void render() override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif

    virtual bool eventCallback(const Event::Event &e) override;

    virtual void setFollowing(Mob *following) override;
    virtual void setEnemy(Mob *enemy) override;
};
```

../src/entity/movableEntity/mob/Player.h

entity/movableEntity/mob/Player.cpp

```
#include "Player.h"

#include "Projectile.h"
#include "Tile.h"

#include "Boomerang.h"
#include "Bow.h"
#include "CrossBow.h"
#include "DarkStaff.h"
#include "FireStaff.h"
#include "FrostStaff.h"
#include "GoldStaff.h"
#include "GoldStaff.h"
#include "EarthStaff.h"
#include "Sling.h"

Player::Player()
{
}

Player::Player(float x, float y)
    : Mob(x, y)
{
}

Player::Player(float x, float y, Level *level)
    : Mob(x, y, level)
{
    changeMaxHealth(1000);
    changeHealth(1000);
    m_Weapons.push_back(new FireStaff());
    m_CurrentWeapon = 0;

    m_Inventory.push_back(new FireStaff());
    m_Inventory.push_back(new FrostStaff());
    m_Inventory.push_back(new DarkStaff());
    m_Inventory.push_back(new GoldStaff());
    m_Inventory.push_back(new EarthStaff());
    m_Inventory.push_back(new Sling());
    m_Inventory.push_back(new Bow());
    m_Inventory.push_back(new Crossbow());
    m_Inventory.push_back(new Boomerang());
}

Player::~Player()
{
}

void Player::update()
{
        if(isInControl)
        {
            Vec2f ratio = {0, 0};
            if(Event::isKeyPressed(GLFW_KEY_W) || Event::isKeyPressed(GLFW_KEY_UP))
                ratio.y += 1.0f;
            if(Event::isKeyPressed(GLFW_KEY_S) || Event::isKeyPressed(GLFW_KEY_DOWN))
                ratio.y -= 1.0f;
            if(Event::isKeyPressed(GLFW_KEY_A) || Event::isKeyPressed(GLFW_KEY_LEFT))
                ratio.x -= 1.0f;
            if(Event::isKeyPressed(GLFW_KEY_D) || Event::isKeyPressed(GLFW_KEY_RIGHT))
                ratio.x += 1.0f;

            if(ratio.x != 0 || ratio.y != 0)
                move(ratio);
            else
```

```
                isMoving = false;

            Mob::update();
        }
}

void Player::render()
{
    Mob::render();
}

bool Player::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::keyInput)
    {
        const Event::KeyboardEvent &ne = static_cast<const Event::KeyboardEvent &>(e);
        if(ne.key == GLFW_KEY_SPACE && (ne.action == GLFW_PRESS || ne.action == GLFW_REPEAT) && m_CurrentWeapon !=
     -1)
        {
            useCurrentWeapon(ne.action == GLFW_REPEAT);
            return true;
        }
    }
    return Mob::eventCallback(e);
}

void Player::setFollowing(Mob *following)
{
    Log::warning("Trying to set player follower!");
}

void Player::setEnemy(Mob *enemy)
{
    Log::warning("Trying to set player enemy!");
}

#ifdef DEBUG
void Player::imGuiRender()
{
    ImGui::Checkbox("Ghost mode", &isGhost);
    ImGui::SliderFloat("Player Speed", &m_Speed, 0.0f, 100.0f);
}
#endif
```

../src/entity/movableEntity/mob/Player.cpp

entity/movableEntity/mob/NPC.h

```
#pragma once

#include "Mob.h"

#include <string>

class NPC : public Mob
{
  public:
    enum class Type
    {
        Follower,
        Enemy
    };

    enum class Race
    {
        Frost,
        Fire,
        Dark
    };

  private:
    std::string m_Name;

    enum class AttackMove
    {
      None,
      Dodge,
      RunAway,
      RunAwayAlongX,
      RunAwayAlongY,
      GoToPoint,
      Attack
    };
    AttackMove m_Attack;

    Vec2f    m_Center;
    Vec2f    m_NextPos;
    bool     m_NextPosActive;
    uint32_t m_TimeSinceMoved;
    uint32_t m_WaitFor;

    bool findingPath;
    bool isRunningAway;

    void generateInventory(Race race);

    void findPath(Vec2f dest, float speed);
    void attack();
    void follow();
    void roam();

    void generateNextPos();

  public:
    NPC();
    NPC(float x, float y, Level *level, Type type);
    NPC(float x, float y, Level *level, Type type, Race race);
    virtual ~NPC() override;

    virtual void update() override;
    virtual void render() override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif

    void goToPointInRoom();

    virtual void setFollowing(Mob *following) override;

    virtual bool eventCallback(const Event::Event &e) override;
};
```

../src/entity/movableEntity/mob/NPC.h

entity/movableEntity/mob/NPC.cpp

```
#include "NPC.h"

#include <thread>
#include <vector>

#include "Application.h"
#include "KeyDefinitions.h"
#include "Level.h"
#include "MessageManager.h"
#include "Player.h"
#include "RandomGen.h"
#include "Utils.h"

#include "Boomerang.h"
#include "Bow.h"
#include "Crossbow.h"
#include "DarkStaff.h"
#include "FireStaff.h"
#include "FrostStaff.h"
#include "Sling.h"

static float getRatioForAttacking(float pos, float ePos, float buffer)
{
    float dist = ePos - pos;
    if(std::fabs(dist) > buffer)
        return dist > 0 ? 1.0f : -1.0f;
    else
        return 0.0f;
}

static Sprite::ID getSpriteID(NPC::Type type, NPC::Race race)
{
    Sprite::ID id;
    switch(type)
    {
    case NPC::Type::Follower:
        id = Sprite::ID::followerFrost;
        break;
    case NPC::Type::Enemy:
        id = Sprite::ID::enemyFrost;
        break;

    default:
        Log::warning("Unknown npc type!");
        id = Sprite::ID::errorID;
        break;
    }

    switch(race)
    {
    case NPC::Race::Frost:
        id += SPRITE_FROST;
        break;
    case NPC::Race::Fire:
        id += SPRITE_FIRE;
        break;
    case NPC::Race::Dark:
        id += SPRITE_DARK;
        break;

    default:
        Log::warning("Unknown npc race!");
        break;
    }

    return id;
}

static Sprite::ID genSpriteID(NPC::Type type)
{
    int r = Random::getNum(0, 2);
    return getSpriteID(type, static_cast<NPC::Race>(r));
}

static NPC::Race getRace(NPC::Type type, Sprite::ID spriteID)
{
    switch(type)
    {
    case NPC::Type::Follower:
        spriteID -= Sprite::ID::followerFrost;
        break;
    case NPC::Type::Enemy:
        spriteID -= Sprite::ID::enemyFrost;
        break;
    default:
        Log::warning("Unknown npc type!");
        break;
    }

    switch(spriteID)
    {
    case SPRITE_FROST:
        return NPC::Race::Frost;
    case SPRITE_FIRE:
        return NPC::Race::Fire;
    case SPRITE_DARK:
        return NPC::Race::Dark;

    default:
        Log::warning("Unknown spriteID given!");
        break;
    }
    return NPC::Race::Frost;
}

NPC::NPC()
    : m_Name("Bob"), m_Attack(AttackMove::None), m_Center({0.0f, 0.0f}), m_NextPos({0.0f, 0.0f}), m_NextPosActive(
    false), m_TimeSinceMoved(0), m_WaitFor(0), findingPath(false), isRunningAway(false)
{
    generateInventory(Race::Fire);
}

NPC::NPC(float x, float y, Level *level, Type type)
    : Mob(x, y, level, genSpriteID(type)), m_Name("Bob"), m_Attack(AttackMove::None), m_Center({x, y}), m_NextPos
    ({0.0f, 0.0f}), m_NextPosActive(false), m_TimeSinceMoved(0), m_WaitFor(0), findingPath(false), isRunningAway(
    false)
```

```
{
    generateInventory(getRace(type, m_SpriteID));
}

NPC::NPC(float x, float y, Level *level, Type type, Race race)
    : Mob(x, y, level, getSpriteID(type, race)), m_Name("Bob"), m_Attack(AttackMove::None), m_Center({x, y}),
    m_NextPos({0.0f, 0.0f}), m_NextPosActive(false), m_TimeSinceMoved(0), m_WaitFor(0), findingPath(false),
    isRunningAway(false)
{
    generateInventory(race);
}

NPC::~NPC()
{
}

void NPC::generateInventory(Race race)
{
    int r = Random::getNum(0, 2);
    if(r == 0)
    {
        int s = Random::getNum(0, 3);
        switch(s)
        {
        case 0:
            pickUp(new Boomerang());
            break;
        case 1:
            pickUp(new Bow());
            break;
        case 2:
            pickUp(new Crossbow());
            break;
        default:
            pickUp(new Sling());
            break;
        }
    }
    else
    {
        switch(race)
        {
        case Race::Frost:
            pickUp(new FrostStaff());
            break;
        case Race::Fire:
            pickUp(new FireStaff());
            break;
        case Race::Dark:
            pickUp(new DarkStaff());
            break;

        default:
            Log::warning("Unknown race when generating weapons!");
            break;
        }
    }
}

void NPC::findPath(Vec2f dest, float speed)
{
    Vec2f start = {x, y};
    if(!m_Level)
        Log::critical("Level is null", LOGINFO);
    std::vector<Vec2f> *path = m_Level->getPath(start, dest, m_CollisionBox);

    if(path->size() == 0 || (path->size() == 1 && distBetweenVec2f({x, y}, path->front()) < speed))
    {
        isMoving = false;
        return;
    }

    float availiableDist = speed;
    while(availiableDist > 0.0f && path->size() > 0)
    {
        float distToNext = distBetweenVec2f({x, y}, path->back());
        Vec2f distVec    = {path->back().x - x, path->back().y - y};
        if(distToNext < availiableDist)
        {
            availiableDist -= distToNext;
            move(distVec.x, distVec.y);
            path->pop_back();
        }
        else
        {
            availiableDist = 0.0f;
            float timesBy  = speed / distToNext;
            move(distVec.x * timesBy, distVec.y * timesBy);
        }
    }

    findingPath = false;
    delete path;
}

void NPC::follow()
{
    if(!m_Following)
    {
        Log::warning("Trying to follow a nullptr!");
        return;
    }
    float xDif        = m_Following->getX() - x;
    float yDif        = m_Following->getY() - y;
    float minDistAway = (TILE_SIZE / 3) * 2;
    if(!findingPath && (xDif < -minDistAway || xDif > minDistAway || yDif < -minDistAway || yDif > minDistAway))
    {
        findingPath = true;
        // std::thread t1(&NPC::findPath, this);
        // t1.detach();
        findPath({m_Following->getX(), m_Following->getY()}, m_Speed);
    }
    else
        isMoving = false;
}

void NPC::roam()
{
    if(!m_NextPosActive)
        generateNextPos();

    float xDif        = m_NextPos.x - x;
    float yDif        = m_NextPos.y - y;
    float minDistAway = TILE_SIZE / 3;

    if(!findingPath && (xDif < -minDistAway || xDif > minDistAway || yDif < -minDistAway || yDif > minDistAway))
    {
        findingPath = true;
        // std::thread t1(&NPC::findPath, this);
        // t1.detach();
        findPath({m_NextPos.x, m_NextPos.y}, m_Speed / 2);
    }
    else
        isMoving = false;

    // This is here just to make sure that if the point cannot be achieved then it picks a new one
    m_TimeSinceMoved++;
    if(m_TimeSinceMoved > m_WaitFor)
        generateNextPos();
}

void NPC::generateNextPos()
{
    m_TimeSinceMoved  = 0;
    m_WaitFor         = Random::getNum(120, 500);
    m_NextPosActive   = true;
    float range       = TILE_SIZE * 2;
    float xPercentage = Random::getNum(-100, 100) / 100.0f;
    float yPercentage = Random::getNum(-100, 100) / 100.0f;
    m_NextPos         = {m_Center.x + range * xPercentage, m_Center.y + range * yPercentage};
}

void NPC::attack()
{
    if(!m_Enemy)
    {
        Log::warning("Trying to attack a nullptr!");
        return;
    }
```

```
// This determines what it should do next
m_TimeSinceMoved++;
if(m_TimeSinceMoved < 0)
    return;

if(m_CurrentWeapon == -1 || distBetweenVec2f({x, y}, {m_Enemy->getX(), m_Enemy->getY()}) < TILE_SIZE &&
m_Attack != AttackMove::GoToPoint)
    m_Attack = AttackMove::RunAway;
else if(getHealth() > 0.8 * getMaxHealth())
    m_Attack = AttackMove::Attack;
else if(m_TimeSinceMoved > m_WaitFor)
{
    if(getStamina() < 0.2 * getMaxStamina() && getHealth() < 0.2 * getMaxHealth())
    {
        m_Attack  = AttackMove::RunAway;    // TODO: Check for potions and use them
        m_WaitFor = Random::getNum(150, 500);
    }
    else if(getStamina() < 0.2 * getMaxStamina())
    {
        m_Attack = AttackMove::Dodge;

        m_WaitFor = Random::getNum(200, 500);
    }
    else if(Random::getNum(0, 3) == 0)
    {
        m_Attack = AttackMove::GoToPoint;
        // Sets the center of the room
        m_Center = {(float) ((int) x / ROOM_PIXEL_SIZE) * ROOM_PIXEL_SIZE + ROOM_PIXEL_SIZE / 2, (float) ((int
) y / ROOM_PIXEL_SIZE) * ROOM_PIXEL_SIZE + ROOM_PIXEL_SIZE / 2};

        m_NextPosActive   = true;
        float range       = TILE_SIZE * 1.5f;
        float xPercentage = Random::getNum(-100, 100) / 100.0f;
        float yPercentage = Random::getNum(-100, 100) / 100.0f;
        m_NextPos         = {m_Center.x + range * xPercentage, m_Center.y + range * yPercentage};

        m_WaitFor = 240;
    }
    else
    {
        m_Attack  = AttackMove::Attack;
        m_WaitFor = Random::getNum(200, 500);
    }
    m_TimeSinceMoved = -10;
}

isMoving = false;

// The NPC attacking
switch(m_Attack)
{
case AttackMove::Dodge:
{
    // TODO: Take stamina into account etc
    float range      = TILE_SIZE * 2;
    auto [dir, proj] = m_Level->getDirOfProjInRange(x, y, range);
    if(proj)
    {
        float xa = 0;
        float ya = 0;
        if(dir == Direction::north || dir == Direction::south)
        {
            float difX = x - m_Enemy->getX();

            xa = difX > 0 ? m_Speed : -m_Speed;
        }
        else
        {
            float difY = y - m_Enemy->getY();

            ya = difY > 0 ? m_Speed : -m_Speed;
        }

        move(xa, ya);
    }
    break;
}

case AttackMove::RunAway:
{
    float difX = x - m_Enemy->getX();
    float difY = y - m_Enemy->getY();
    if(std::fabs(difX) < 10.0f)
        difX += difX > 0 ? 20.0f : -20.0f;
    if(std::fabs(difY) < 10.0f)
        difY += difY > 0 ? 20.0f : -20.0f;
    if(canMove({difX, difY}))
        move({difX, difY});
    else
        goToPointInRoom();
    break;
}

case AttackMove::RunAwayAlongX:
{
    float difX = x - m_Enemy->getX();
    if(canMove({difX, 0.0f}))
        move({difX, 0.0f});
    else
        goToPointInRoom();
    break;
}

case AttackMove::RunAwayAlongY:
{
    float difY = y - m_Enemy->getY();
    if(canMove({0.0f, difY}))
        move({0.0f, difY});
    else
        goToPointInRoom();
    break;
}

case AttackMove::GoToPoint:
{
    float xDif        = m_NextPos.x - x;
    float yDif        = m_NextPos.y - y;
    float minDistAway = TILE_SIZE / 3;

    if(!findingPath && (xDif < -minDistAway || xDif > minDistAway || yDif < -minDistAway || yDif > minDistAway
))
    {
        findingPath = true;   // TODO: Get rid of this variable
        findPath({m_NextPos.x, m_NextPos.y}, m_Speed * (getStamina() / getMaxStamina()) / 2);
    }
    else
        m_Attack = AttackMove::None;

    break;
}

default:   // Attack
{
    // TODO: Check if friend is in line of fire
    float buffer = TILE_SIZE / 4.0f;
    Vec2f ratio  = {getRatioForAttacking(x, m_Enemy->getX(), buffer), getRatioForAttacking(y, m_Enemy->getY(),
 buffer)};

    if((ratio.y == 0.0f || ratio.x == 0.0f))   // Attacks if can
    {
        if(m_Weapons[m_CurrentWeapon]->canUse())
        {
            if(ratio.x == 0.0f)
            {

                if(y > m_Enemy->getY())
                    m_Dir = Direction::south;
                else
                    m_Dir = Direction::north;
            }
            else
            {
                if(x > m_Enemy->getX())
                    m_Dir = Direction::west;
                else
                    m_Dir = Direction::east;
            }

            useCurrentWeapon(true);
        }
        }
        else
        {
            float distX = std::fabs(ratio.x - m_Enemy->getX());
            float distY = std::fabs(ratio.y - m_Enemy->getX());

            if(distX > distY)
            {
                if(distX < TILE_SIZE)   // This checks that the distance is within a given range and corrects accordingly
                {
                    m_TimeSinceMoved = 0;
                    m_WaitFor        = 60;
                    m_Attack         = AttackMove::RunAwayAlongX;
                }
                else if(std::fabs(ratio.x - m_Enemy->getX()) < 2 * TILE_SIZE)
                    ratio.x = 0.0f;
            }
            else
            {
                if(distY < TILE_SIZE)
                {
                    m_TimeSinceMoved = 0;
                    m_WaitFor        = 60;
                    m_Attack         = AttackMove::RunAwayAlongX;
                }
                else if(distY < 2 * TILE_SIZE)
                    ratio.y = 0.0f;
            }

            if(m_Attack == AttackMove::Attack)
                move(ratio);
        }
        break;
    }
    }
}

void NPC::update()
{
    if(m_Enemy && (int) m_Enemy->getX() / (TILE_SIZE * ROOM_SIZE) == (int) x / (TILE_SIZE * ROOM_SIZE) && (int) m_Enemy->getY() / (TILE_SIZE * ROOM_SIZE) == (int) y / (TILE_SIZE * ROOM_SIZE))
        attack();
    else
    {
        if(m_Attack != AttackMove::None)
            m_Attack = AttackMove::None;

        if(m_Following && !findingPath)
            follow();
        else
            roam();
    }

    Mob::update();
}

void NPC::render()
{
    Mob::render();
}

#ifdef DEBUG
void NPC::imGuiRender()
{
}

#endif

void NPC::goToPointInRoom()
{

    m_Attack         = AttackMove::GoToPoint;
    m_TimeSinceMoved = 0;
    // Sets the center of the room
    m_Center = {(float) ((int) x / ROOM_PIXEL_SIZE) * ROOM_PIXEL_SIZE + ROOM_PIXEL_SIZE / 2, (float) ((int) y / ROOM_PIXEL_SIZE) * ROOM_PIXEL_SIZE + ROOM_PIXEL_SIZE / 2};

    m_NextPosActive   = true;
    float range       = TILE_SIZE * 1.5f;
    float xPercentage = Random::getNum(-100, 100) / 100.0f;
    float yPercentage = Random::getNum(-100, 100) / 100.0f;
    m_NextPos         = {m_Center.x + range * xPercentage, m_Center.y + range * yPercentage};
}

void NPC::setFollowing(Mob *following)
{
    m_Center = {x, y};
    Mob::setFollowing(following);
}

bool NPC::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mouseClicked)
    {
        const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);

        Vec2f convPos = Application::getCamera()->convertWindowToLevel(ne.pos);

        Player *player = m_Level->getPlayer();
        if(doesPointIntersectWithBox(convPos, {x, y}, {{-width / 2, -height / 2}, {width / 2, height / 2}}) &&
    distBetweenVec2f({player->getX(), player->getY() - player->getWidth() / 2}, {x, y}) < 5.0f * TILE_SIZE)
        {
            if(!m_Enemy && !m_Following)
            {
                Event::ChangeGUIActiveLayer e(InGameGUILayer::npcInteraction);
                Application::callEvent(e, true);

                MessageManager::sendMessage("NPC: Can I follow you?", MessageManager::Priority::Low);

                return true;
            }
            else if(m_Following == m_Level->getPlayer())
            {
                Event::ChangeGUIActiveLayer e1(InGameGUILayer::npcInventory);
                Application::callEvent(e1, true);

                Event::ChestOpenedEvent e2(&m_Inventory, nullptr, GUIInventoryIDCode::inventory);
                Application::callEvent(e2, true);

                Event::ChestOpenedEvent e3(&m_Weapons, &m_CurrentWeapon, GUIInventoryIDCode::weapons);
                Application::callEvent(e3, true);
                return true;
            }

            return false;
        }
    }
    else if(e.getType() == Event::EventType::playerResponse && !m_Following && !m_Enemy)
    {
        const Event::PlayerResponse &ne = static_cast<const Event::PlayerResponse &>(e);

        if(ne.response == Event::PlayerResponse::Response::reject)
        {
            // TODO: Have this have a random change to attack the player
            int r = Random::getNum(0, 4);   // TODO: Base this upon stats?

            if(r == 1)
            {
                m_Enemy = m_Level->getPlayer();

                Event::ShowAltTileEvent e(true);
                Application::callEvent(e);

                MessageManager::sendMessage("NPC: How rude!? I will kill you now!", MessageManager::Priority::
    Medium);
            }
            else
                MessageManager::sendMessage("NPC: Oh okay, fine be like that.", MessageManager::Priority::Low);
        }
        else
        {
            if(m_Level->getPlayer()->addFollower(this))
            {
                MessageManager::sendMessage("NPC: Yay! I know we will be best buds!", MessageManager::Priority::
    Low);
                m_Following = m_Level->getPlayer();
            }
            else
                MessageManager::sendMessage("NPC: Looks like you already have a follower", MessageManager::
```

```
    Priority::Medium);
        }

        Event::ChangeGUIActiveLayer e(InGameGUILayer::overlay);
        Application::callEvent(e, true);

        return true;
    }
    else if(e.getType() == Event::EventType::mazeMovedEvent)
    {
        const Event::MazeMovedEvent &ne = static_cast<const Event::MazeMovedEvent &>(e);
        m_Center.x += ne.changeX;
        m_Center.y += ne.changeY;
        if(m_NextPosActive)
        {
            m_NextPos.x += ne.changeX;
            m_NextPos.y += ne.changeY;
        }
    }
    return Mob::eventCallback(e);
}
```

../src/entity/movableEntity/mob/NPC.cpp

3.11.3 Projectile

entity/movableEntity/projectile/Projectile.h

```
#pragma once

#include <functional>

#include "MovableEntity.h"

class Projectile : public MovableEntity
{
  public:
    enum class Type
    {
        Arrow,
        Boomerang,
        Dark,
        Fire,
        Frost,
        Gold,
        Nature,
        Rock
    };

  protected:
    Vec2f    m_StartPos;
    float    m_MaxDistance;
    float    m_Damage;
    Entity *spawner;
    bool     hasCollided;
    double   m_Rotation;
    double   m_RotationSpeed;

    std::function<void(float, float, Direction, Level *)> m_CollisionFunction;

  public:
    Projectile(float startX, float startY, float damage, Direction dir, Entity *spawner, Level *level, Type type);
    Projectile(float startX, float startY, float size, float damage, Direction dir, Entity *spawner, Level *level)
    ;
    Projectile(float startX, float startY, float size, float damage, float speed, Direction dir, Entity *spawner,
    Level *level, CollisionBox box);
    Projectile(float startX, float startY, float size, float maxDistance, float damage, float speed, Direction dir
    , Entity *spawner, Level *level, CollisionBox box, std::function<void(float, float, Direction, Level *)>
    collisionFunc);
    virtual ~Projectile() override {}

    virtual void update() override;
    virtual void render() override;
    virtual bool eventCallback(const Event::Event &e) override;

    virtual void changeX(float changeBy) override;
    virtual void changeY(float changeBy) override;
    virtual bool deleteMe() override { return hasCollided; }

    CollisionBox getEntityBox() { return {{20, 20}, m_CollisionBox.upperBound}; }
};
```

../src/entity/movableEntity/projectile/Projectile.h

entity/movableEntity/projectile/Projectile.cpp

```
#include "Projectile.h"

#include "Entity.h"
#include "KeyDefinitions.h"
#include "Level.h"
#include "ParticleSpawner.h"
#include "Sprite.h"
#include "RandomGen.h"

#define defaultBox            \
    {                         \
        {45, 45}, { 60, 55 } \
    }

Projectile::Projectile(float startX, float startY, float damage, Direction dir, Entity *spawner, Level *level,
    Type type)
    : MovableEntity(startX, startY, TILE_SIZE / 2, 7.0f, dir, defaultBox, level, Sprite::ID::errorID), m_StartPos
    ({startX, startY}), m_MaxDistance(7 * TILE_SIZE), m_Damage(damage), spawner(spawner), hasCollided(false),
    m_Rotation(directionToRotation(dir)), m_RotationSpeed(0.0f)
{
    switch(type)
    {
    case Type::Arrow:
        m_SpriteID          = Sprite::ID::projectileArrow;
        m_Speed             = 15.0f;
        m_MaxDistance       = 10 * TILE_SIZE;
        m_CollisionBox      = {{45, 45}, {50, 50}};
        m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
            float xMinSpeed = -10.0f;
            float xMaxSpeed = 10.0f;
            float yMinSpeed = -10.0f;
            float yMaxSpeed = 10.0f;
            if(dir == Direction::north)
            {
                yMaxSpeed = 2.5f;
                y += 40;
            }
            else if(dir == Direction::south)
            {
                yMinSpeed = -2.5f;
                y -= 40;
            }
            else if(dir == Direction::east)
            {
                xMaxSpeed = 2.5f;
                x += 30;
            }
            else
            {
                xMinSpeed = -2.5f;
                x -= 50;
            }

            uint16_t  spawnerLifetime = 5;
            uint16_t  spawnRate       = 2;
            uint16_t  minLife         = 5;
            uint16_t  maxLife         = 15;
            float     minSize         = 2.0f;
            float     maxSize         = 7.0f;
            uint16_t  groupSize       = 2;
            glm::vec4 colour          = {0.471, 0.518f, 0.671f, 1.0f};

            level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
    xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
        };

        break;
    case Type::Boomerang:
        m_SpriteID          = Sprite::ID::weaponBoomerang;
        m_Speed             = 12.5f;
        m_MaxDistance       = 10 * TILE_SIZE;
        m_RotationSpeed     = Random::getNum(3, 7) / 10.0f;
        m_MaxDistance       = 4 * TILE_SIZE;
        m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
            float xMinSpeed = -6.0f;
        float xMaxSpeed = 6.0f;
        float yMinSpeed = -6.0f;
        float yMaxSpeed = 6.0f;
        if(dir == Direction::north)
        {
            yMaxSpeed = 0.5f;
            y += 40;
        }
        else if(dir == Direction::south)
        {
            yMinSpeed = -0.5f;
            y -= 40;
        }
        else if(dir == Direction::east)
        {
            xMaxSpeed = 0.5f;
            x += 30;
        }
        else
        {
            xMinSpeed = -0.5f;
            x -= 50;
        }

        uint16_t  spawnerLifetime = 6;
        uint16_t  spawnRate       = 1;
        uint16_t  minLife         = 2;
        uint16_t  maxLife         = 10;
        float     minSize         = 4.0f;
        float     maxSize         = 15.0f;
        uint16_t  groupSize       = 2;
        glm::vec4 colour          = {1.0f, 0.749f, 0.212f, 1.0f};

        level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
    };

    break;
case Type::Dark:
    m_SpriteID          = Sprite::ID::projectileDark;
    m_Speed             = 10.0f;
    m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
        float xMinSpeed = -2.0f;
        float xMaxSpeed = 2.0f;
        float yMinSpeed = -2.0f;
        float yMaxSpeed = 2.0f;
        if(dir == Direction::north)
        {
            yMaxSpeed = 0.0f;
            y += 40;
        }
        else if(dir == Direction::south)
        {
            yMinSpeed = 0.0f;
            y -= 40;
        }
        else if(dir == Direction::east)
        {
            xMaxSpeed = 0.0f;
            x += 30;
        }
        else
        {
            xMinSpeed = 0.0f;
            x -= 50;
        }

        uint16_t  spawnerLifetime = 5;
        uint16_t  spawnRate       = 1;
        uint16_t  minLife         = 10;
        uint16_t  maxLife         = 30;
        float     minSize         = 4.0f;
        float     maxSize         = 10.0f;
        uint16_t  groupSize       = 4;
        glm::vec4 colour          = {0.216f, 0.216f, 0.216f, 1.0f};

        level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
    };
    break;
case Type::Fire:
    m_SpriteID          = Sprite::ID::projectileFire;
    m_Speed             = 12.5f;
    m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
        float xMinSpeed = -8.0f;
        float xMaxSpeed = 8.0f;
        float yMinSpeed = -8.0f;
        float yMaxSpeed = 8.0f;
        if(dir == Direction::north)
        {
            yMaxSpeed = 0.5f;
            y += 40;
        }
        else if(dir == Direction::south)
        {
            yMinSpeed = -0.5f;
            y -= 40;
        }
        else if(dir == Direction::east)
        {
            xMaxSpeed = 0.5f;
            x += 30;
        }
        else
        {
            xMinSpeed = -0.5f;
            x -= 50;
        }

        uint16_t  spawnerLifetime = 5;
        uint16_t  spawnRate       = 1;
        uint16_t  minLife         = 5;
        uint16_t  maxLife         = 15;
        float     minSize         = 7.0f;
        float     maxSize         = 15.0f;
        uint16_t  groupSize       = 3;
        glm::vec4 colour          = {0.929f, 0.541f, 0.0f, 1.0f};

        level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
    };

    break;
case Type::Frost:
    m_SpriteID          = Sprite::ID::projectileFrost;
    m_Speed             = 9.0f;
    m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
        float xMinSpeed = -4.0f;
        float xMaxSpeed = 4.0f;
        float yMinSpeed = -4.0f;
        float yMaxSpeed = 4.0f;
        if(dir == Direction::north)
        {
            yMaxSpeed = 0.0f;
            y += 40;
        }
        else if(dir == Direction::south)
        {
            yMinSpeed = 0.0f;
            y -= 40;
        }
        else if(dir == Direction::east)
        {
            xMaxSpeed = 0.0f;
            x += 30;
        }
        else
        {
            xMinSpeed = 0.0f;
            x -= 50;
        }

        uint16_t  spawnerLifetime = 5;
        uint16_t  spawnRate       = 1;
        uint16_t  minLife         = 5;
        uint16_t  maxLife         = 20;
        float     minSize         = 5.0f;
        float     maxSize         = 12.0f;
        uint16_t  groupSize       = 4;
        glm::vec4 colour          = {0.447f, 0.773f, 0.835f, 1.0f};

        level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
    };

    break;
case Type::Gold:
    m_SpriteID          = Sprite::ID::projectileGold;
    m_Speed             = 15.0f;
    m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
        float xMinSpeed = -10.0f;
        float xMaxSpeed = 10.0f;
        float yMinSpeed = -10.0f;
        float yMaxSpeed = 10.0f;
        if(dir == Direction::north)
        {
            yMaxSpeed = 4.0f;
            y += 40;
        }
        else if(dir == Direction::south)
        {
            yMinSpeed = -4.0f;
            y -= 40;
        }
        else if(dir == Direction::east)
        {
            xMaxSpeed = 4.0f;
            x += 30;
        }
        else
        {
            xMinSpeed = -4.0f;
            x -= 50;
        }

        uint16_t  spawnerLifetime = 5;
        uint16_t  spawnRate       = 1;
        uint16_t  minLife         = 3;
        uint16_t  maxLife         = 10;
        float     minSize         = 3.0f;
        float     maxSize         = 9.0f;
        uint16_t  groupSize       = 4;
        glm::vec4 colour          = {0.886f, 0.773f, 0.478f, 1.0f};

        level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
    };

    break;
case Type::Nature:
    m_SpriteID          = Sprite::ID::projectileNature;
    m_Speed             = 9.0f;
    m_CollisionFunction = [](float x, float y, Direction dir, Level *level) {
        float xMinSpeed = -3.0f;
        float xMaxSpeed = 3.0f;
        float yMinSpeed = -3.0f;
        float yMaxSpeed = 3.0f;
        if(dir == Direction::north)
        {
            yMaxSpeed = 0.5f;
            y += 40;
        }
        else if(dir == Direction::south)
        {
            yMinSpeed = -0.5f;
            y -= 40;
        }
        else if(dir == Direction::east)
        {
            xMaxSpeed = 0.5f;
            x += 30;
        }
        else
        {
            xMinSpeed = -0.5f;
            x -= 50;
        }

        uint16_t  spawnerLifetime = 6;
```

```
            uint16_t  spawnRate       = 2;
            uint16_t  minLife         = 2;
            uint16_t  maxLife         = 10;
            float     minSize         = 10.0f;
            float     maxSize         = 20.0f;
            uint16_t  groupSize       = 2;
            glm::vec4 colour          = {0.682f, 0.867f, 0.502f, 1.0f};

            level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
    xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
        };

        break;
    case Type::Rock:
        m_SpriteID          = Sprite::ID::projectileRock;
        m_Speed             = 9.0f;
        width               = Random::getNum(2, 6) * TILE_SIZE / 20;
        height              = Random::getNum(2, 6) * TILE_SIZE / 20;
        m_Rotation          = Random::getNum(-30, 30) / 10.0f;
        m_RotationSpeed     = Random::getNum(-5, 5) / 20.0f;
        m_MaxDistance       = 3 * TILE_SIZE;
        m_CollisionFunction = [this](float x, float y, Direction dir, Level *level) {
            float xMinSpeed = -6.0f;
            float xMaxSpeed = 6.0f;
            float yMinSpeed = -6.0f;
            float yMaxSpeed = 6.0f;
            if(dir == Direction::north)
            {
                yMaxSpeed = 0.5f;
                y += 40;
            }
            else if(dir == Direction::south)
            {
                yMinSpeed = -0.5f;
                y -= 40;
            }
            else if(dir == Direction::east)
            {
                xMaxSpeed = 0.5f;
                x += 30;
            }
            else
            {
                xMinSpeed = -0.5f;
                x -= 50;
            }

            uint16_t  size            = width > height ? width : height;

            uint16_t  spawnerLifetime = 5;
            uint16_t  spawnRate       = 1;
            uint16_t  minLife         = 5;
            uint16_t  maxLife         = 15;
            float     minSize         = size / 12;
            float     maxSize         = size / 3;
            uint16_t  groupSize       = 4;
            glm::vec4 colour          = {0.243f, 0.216f, 0.361f, 1.0f};

            level->addSpawner(new ParticleSpawner(x, y, level, spawnerLifetime, spawnRate, minLife, maxLife,
    xMinSpeed, xMaxSpeed, yMinSpeed, yMaxSpeed, minSize, maxSize, groupSize, colour));
        };

        break;

    default:
        Log::warning("Unknown projectile!");
        m_Speed        = 7.0f;
        m_SpriteID     = Sprite::ID::errorID;
        m_CollisionBox = defaultBox;
        width          = TILE_SIZE / 2;
        height         = TILE_SIZE / 2;
        m_MaxDistance  = 7 * TILE_SIZE;

        break;
    }
}

Projectile::Projectile(float startX, float startY, float size, float damage, Direction dir, Entity *spawner, Level
     *level)
    : MovableEntity(startX, startY, size, 7.0f, dir, defaultBox, level, Sprite::ID::projectileFire), m_StartPos({
```

```
    startX, startY}), m_MaxDistance(7 * TILE_SIZE), m_Damage(damage), spawner(spawner), hasCollided(false),
    m_Rotation(directionToRotation(dir)), m_RotationSpeed(0.0f)
{
}

Projectile::Projectile(float startX, float startY, float size, float damage, float speed, Direction dir, Entity *
    spawner, Level *level, CollisionBox box)
    : MovableEntity(startX, startY, size, speed, dir, box, level, Sprite::ID::projectileFire), m_StartPos({startX,
     startY}), m_MaxDistance(7 * TILE_SIZE), m_Damage(damage), spawner(spawner), hasCollided(false), m_Rotation(
    directionToRotation(dir)), m_RotationSpeed(0.0f)
{
}

Projectile::Projectile(float startX, float startY, float size, float maxDistance, float damage, float speed,
    Direction dir, Entity *spawner, Level *level, CollisionBox box, std::function<void(float, float, Direction,
    Level *)> collisionFunc)
    : MovableEntity(startX, startY, size, speed, dir, box, level, Sprite::ID::projectileFire), m_StartPos({startX,
     startY}), m_MaxDistance(maxDistance), m_Damage(damage), spawner(spawner), hasCollided(false),
    m_CollisionFunction(collisionFunc), m_Rotation(directionToRotation(dir)), m_RotationSpeed(0.0f)
{
}

void Projectile::update()
{
    if(!hasCollided)
    {
        m_Rotation += m_RotationSpeed;

        if(distBetweenVec2f({x, y}, m_StartPos) > m_MaxDistance)
            hasCollided = true;
        float xs = 0;
        float ys = 0;
        if(m_Dir == Direction::north)
            ys = m_Speed;
        else if(m_Dir == Direction::south)
            ys = -m_Speed;
        else if(m_Dir == Direction::east)
            xs = m_Speed;
        else
            xs = -m_Speed;
        Entity *colE = m_Level->entityCollisionDetection(x + xs, y + ys, getEntityBox());
        if(!isGhost && (m_Level->collisionDetection(x + xs, y + ys, m_CollisionBox) || (colE != nullptr && colE !=
     spawner)))
        {
            Mob *mSpawner = dynamic_cast<Mob *>(spawner);
            if(colE)
            {
                Mob *mob = dynamic_cast<Mob *>(colE);
                if(mob)
                    mob->dealDamage(m_Damage);
                if(mSpawner)
                    mSpawner->hasHitTarget(m_Damage);
            }
            else if(mSpawner)
                mSpawner->hasMissedTarget();
            m_CollisionFunction(x, y, m_Dir, m_Level);
            hasCollided = true;
        }
        else
        {
            isMoving = true;
            x += xs;
            y += ys;
        }
    }
}

bool Projectile::eventCallback(const Event::Event &e)
{
    if(e.getType() == Event::EventType::mobDied)
    {
        const Event::MobDied &ne = static_cast<const Event::MobDied &>(e);
        if(ne.mob == spawner)
        {
            spawner     = nullptr;
            hasCollided = true;
        }
    }
    return MovableEntity::eventCallback(e);
}
```

```
void Projectile::render()
{
    uint8_t layer = 4;
    Render::sprite(x, y, m_Rotation, width, height, m_SpriteID, layer);
}

void Projectile::changeX(float changeBy)
{
    x += changeBy;
    m_StartPos.x += changeBy;
}
void Projectile::changeY(float changeBy)
{
    y += changeBy;
    m_StartPos.y += changeBy;
}
```

../src/entity/movableEntity/projectile/Projectile.cpp

3.11.4 Object

entity/objects/WorldItem.h

```
#pragma once

#include "Button.h"
#include "Entity.h"
#include "Item.h"
#include "Tile.h"

class WorldItem : public Entity
{
  protected:
    Item *m_Item;
    Button::State m_State;

  public:
    WorldItem(Item *item);
    WorldItem(float x, float y, float size, Item *item);
    WorldItem(float x, float y, float size, Level *level, Item *item);
    virtual ~WorldItem() override;

    virtual void render() override;

    virtual void update() override;
    virtual bool eventCallback(const Event::Event &e) override;
#ifdef DEBUG
    virtual void imGuiRender() override;
#endif

    Item *pickUp();

    virtual bool deleteMe() override;

    virtual bool hasCollidedWith(float xs, float ys, CollisionBox box) override { return false; }

    Item *peak();
};
```

../src/entity/objects/WorldItem.h

entity/objects/WorldItem.cpp

```
#include "WorldItem.h"

#include "Application.h"
#include "KeyDefinitions.h"
#include "Level.h"
#include "MessageManager.h"
#include "Player.h"

WorldItem::WorldItem(Item *item)
    : Entity(0.0f, 0.0f, TILE_SIZE / 2, nullptr, item->getSpriteID()), m_Item(item), m_State(Button::State::None)
    {}

WorldItem::WorldItem(float x, float y, float size, Item *item)
    : Entity(x, y, size, nullptr, item->getSpriteID()), m_Item(item), m_State(Button::State::None) {}

WorldItem::WorldItem(float x, float y, float size, Level *level, Item *item)
```

```
    : Entity(x, y, size, level, item->getSpriteID()), m_Item(item), m_State(Button::State::None) {}

WorldItem::~WorldItem()
{
    if(m_Item)
        delete m_Item;
}

void WorldItem::render()
{
    if(m_Item)
    {
        m_Item->render(x, y, 0.0f, width, 2);
        if(m_State == Button::State::Hover && !Application::getIsPaused())
        {
            float        scale    = 35.0f;
            Vec2f        mousePos = Application::getCamera()->convertWindowToLevel(Event::getMousePos());

            uint8_t layer = 6;

            Render::hoverText(*m_Item->getName(), mousePos.x, mousePos.y, scale, {1.0f, 1.0f, 1.0f, 1.0f}, {0.3f,
    0.3f, 0.3f, 0.7f}, layer);
        }
    }
}

void WorldItem::update()
{
    if(m_Item)
    {
        Vec2f mousePos = Application::getCamera()->convertWindowToLevel(Event::getMousePos());

        if(mousePos.x > x + m_CollisionBox.lowerBound.x && mousePos.x < x + m_CollisionBox.upperBound.x &&
    mousePos.y > y + m_CollisionBox.lowerBound.y && mousePos.y < y + m_CollisionBox.upperBound.y)
            m_State = Button::State::Hover;
        else
            m_State = Button::State::None;
    }
}
bool WorldItem::eventCallback(const Event::Event &e)
{
    if(!m_Item)
        return false;
    if(e.getType() == Event::EventType::mouseClicked && m_Level)
    {
        const Event::MouseClickedEvent &ne = static_cast<const Event::MouseClickedEvent &>(e);

        Vec2f convPos = Application::getCamera()->convertWindowToLevel(ne.pos);

        Player *player = m_Level->getPlayer();
        if(doesIntersectWith(convPos))
        {
            if(distBetweenVec2f({player->getX(), player->getY() - player->getWidth() / 2}, {x, y}) < 1.5f *
    TILE_SIZE)
            {
                if(player->pickUp(m_Item))
                {
                    Log::info("Picked up");
                    m_Item = nullptr;
                }
                else
                    MessageManager::sendMessage("Inventory full!", MessageManager::Priority::High);
            }
            else
                MessageManager::sendMessage("The item is too far away!", MessageManager::Priority::Medium);

            return true;
        }
    }
    return Entity::eventCallback(e);
}
#ifdef DEBUG
void WorldItem::imGuiRender()
{
}
#endif

Item *WorldItem::pickUp()
{
    Item *temp = m_Item;
    m_Item      = nullptr;
    return temp;
}

bool WorldItem::deleteMe()
{
    return !m_Item;
}

Item *WorldItem::peak() { return m_Item; }
```

../src/entity/objects/WorldItem.cpp

3.12 Headers

These headers are used because they allow me to precompile the headers so that it takes a shorter time to compile the whole project

headers/Core.h

```
#pragma once

#include "glfwGlew.h"

#include <GLM.h>

#include "ImGui.h"

#include <iostream>
```

../src/headers/Core.h

headers/glfwGlew.h

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
```

../src/headers/glfwGlew.h

headers/GLM.h

```
#pragma once

#include "glm/glm.hpp"
#include "glm/gtc/matrix_transform.hpp"
```

../src/headers/GLM.h

headers/ImGui.h

```
#pragma once

#include "imgui/imgui.h"

#include "imgui/examples/imgui_impl_glfw.h"
#include "imgui/examples/imgui_impl_opengl3.h"
```

../src/headers/ImGui.h

headers/KeyDefinitions.h

```
#define TILE_SIZE        100
#define ROOM_SIZE        7
#define ROOM_PIXEL_SIZE (ROOM_SIZE * TILE_SIZE)

#define MAZE_SIZE 11

#define ANIMATION_FRAMES 2

#define DEFAULT_INVENTORY_SIZE 20
```

../src/headers/KeyDefinitions.h

headers/LogHeaders.h

```
#include <chrono>
#include <ctime>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <thread>
```

../src/headers/LogHeaders.h

headers/MazeHeaders.h

```
#include <memory>
#include <thread>
#include <tuple>
#include <vector>
```

../src/headers/MazeHeaders.h

4 Testing

4.1 Hello

Yo yo yo

5 Evaluation

5.1 Hello

Yo yo yo