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 staircase leading downwards.

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?

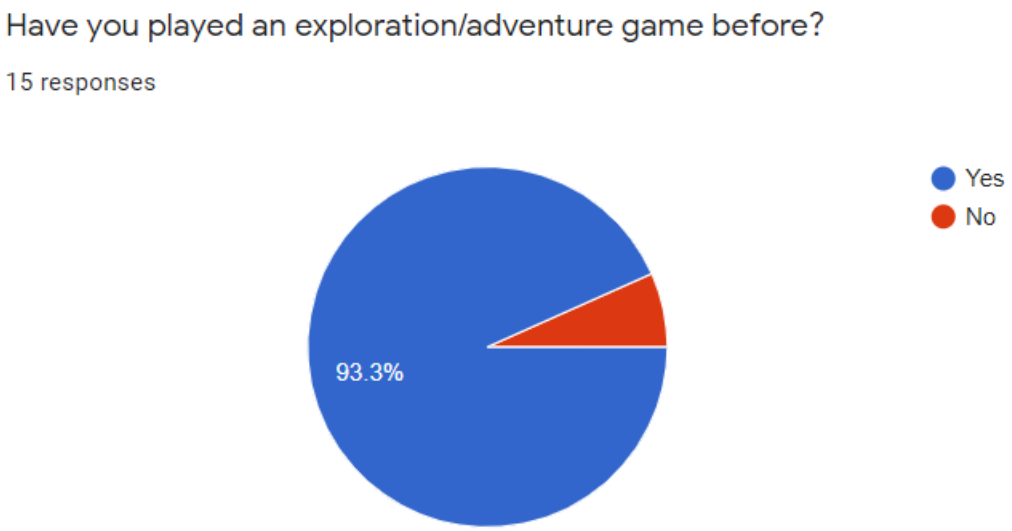


Figure 2: Responses from survey showing most people have played an exploration game

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

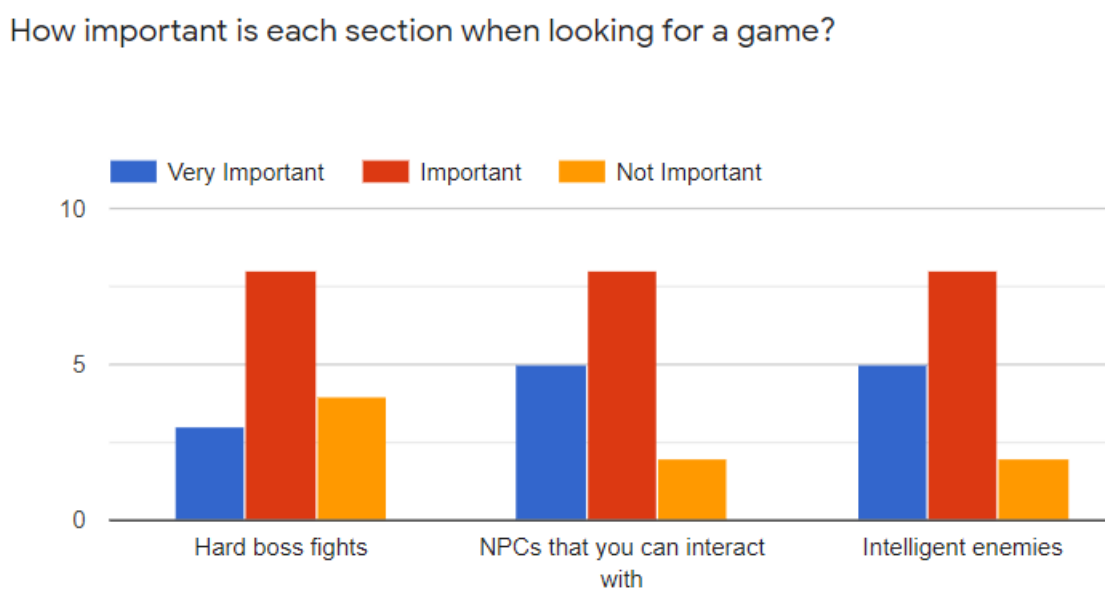


Figure 3: Responses from the story showing that it is equally important to have intelligent enemies and NPCs you can interact with

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

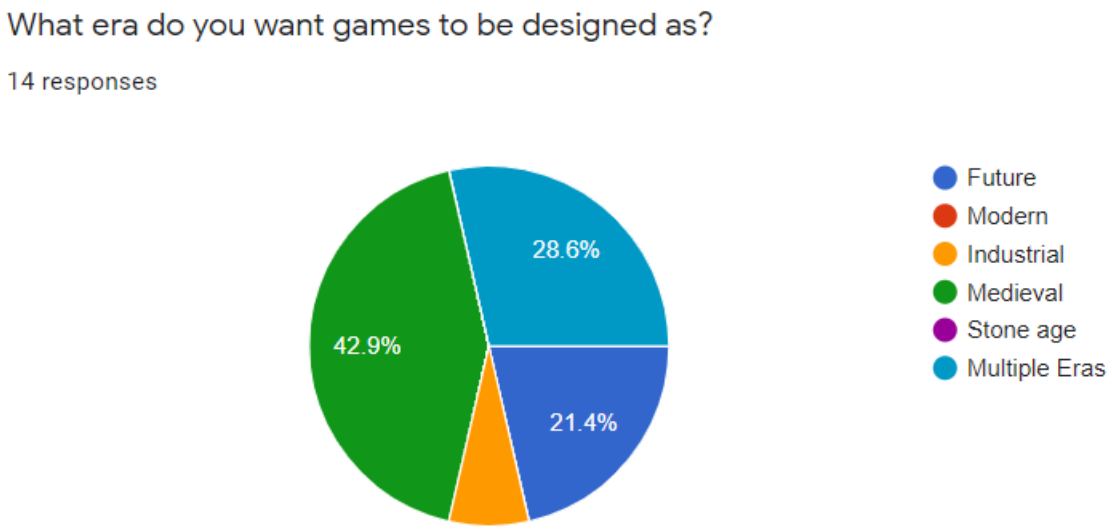


Figure 4: Responses from survey showing that most people like a Medieval design

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

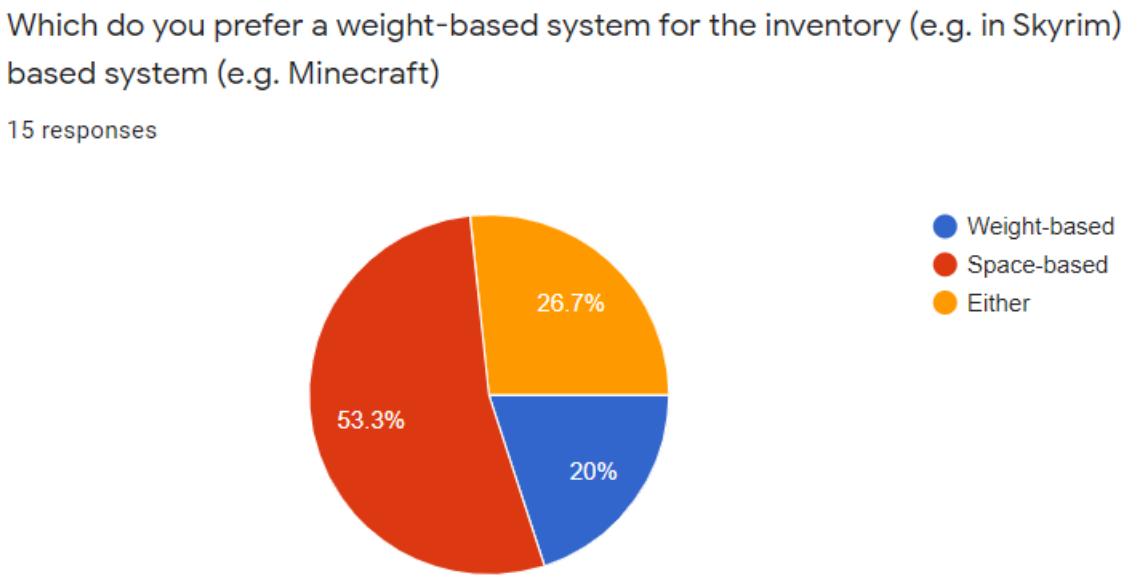


Figure 5: Responses from survey showing that most people like a weight-based system in a game

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

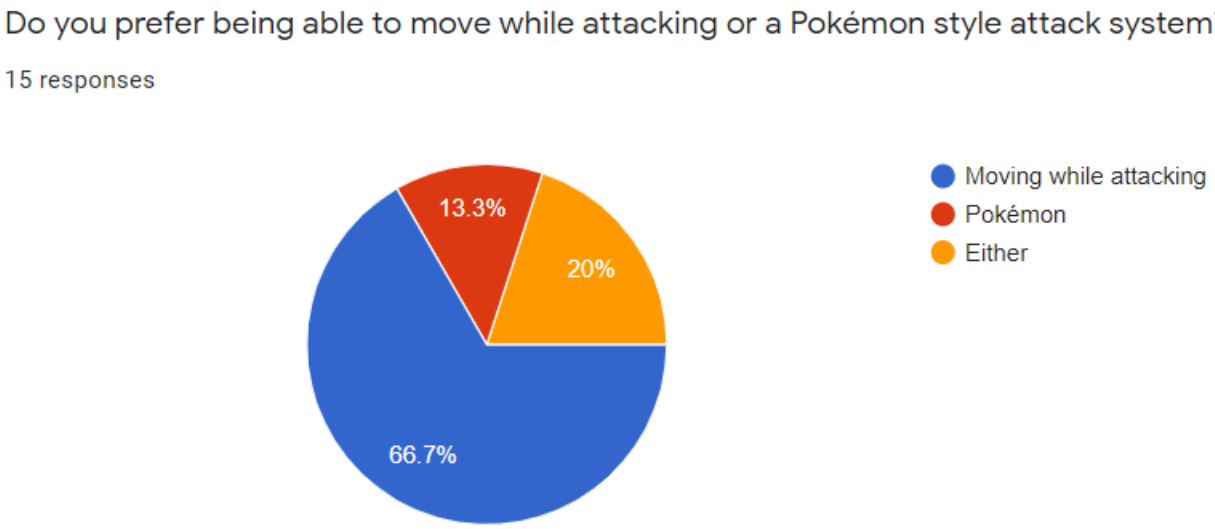


Figure 6: Responses from survey showing that the attacking system should allow you to still control the player

- Have you played "The Binding of Isaac"?

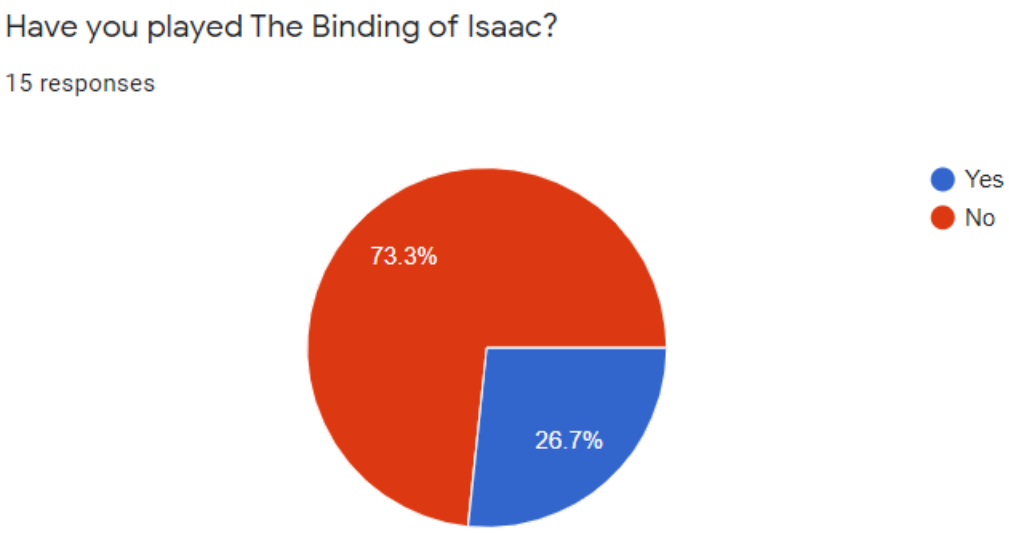


Figure 7: Responses from survey showing that most people in this survey had not played The Binding of Isaac

  - 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.

When asked what they would improve about the game, a few people responded saying:

- More routes down
- A good run is greatly depend on loot found in the first couple of floors.

From this, it seems that the reliance of chance for loot needs to be carefully managed. So, to combat this I have decided to change loot up between the levels, meaning that you cannot get overpowered loot on the first levels. Also I have decided to add a leveling system that allows you to upgrade your stats to allow more ways to play the game. Furthermore, I will need to create multiple ways to get down to different levels, for examples stairs, or a hidden passage that leads you to a treasure room on the next level.

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 a follower 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.
  - Create each character in the NPCs section, with them stored in the maze level, each of them with a different skin.
- 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. Implementing a temporary, basic inventory menu.
  - 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.
  - Add an enemy class that can be rendered onto the maze.
  - 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 effects.
  - Add an experience counter, which will allow the player to increase their stats
- Create different rooms that can be found in the maze.
  - Create the each room and its effects in the "Room" section in the "Design Outline".
  - This will include creating a chest that randomly generates items inside, which the player can pick up.
  - Update the addRoom function to allow the room to be randomised, adding a random room in the place.
- 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
  - Allow the game to be saved and loaded from the main menu (saving them in custom files)
  - Add more stats to the mobs, which results in different effects to your damage and the number of followers you can have.

2 Documented Design

2.1 Overview

2.2 Maze Generation

2.2.1 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.

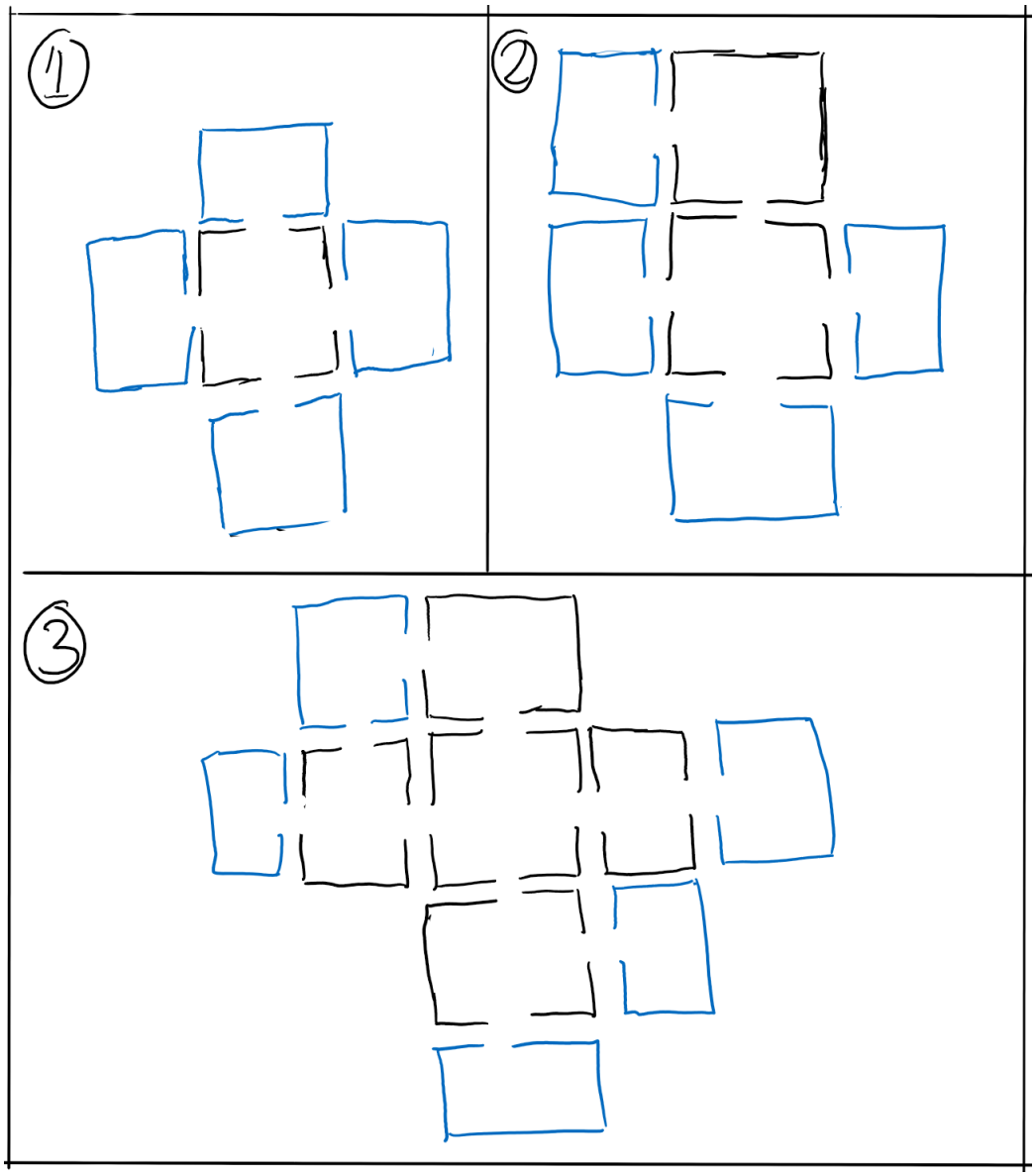


Figure 8: Steps followed by the maze generation

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.

```
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 openned
        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):
```

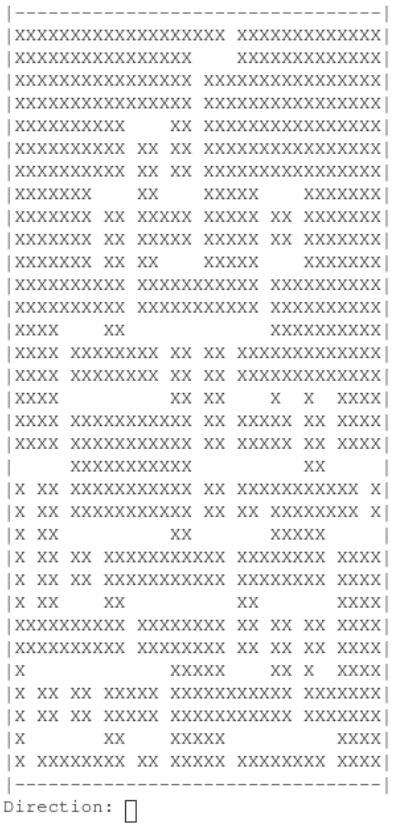
```
    newCurrent = [(x, BOARDSIZE - 1) for x in current]
    for i in range(BOARDSIZE):
        del board[i][0]
        board[i].append(None)

    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

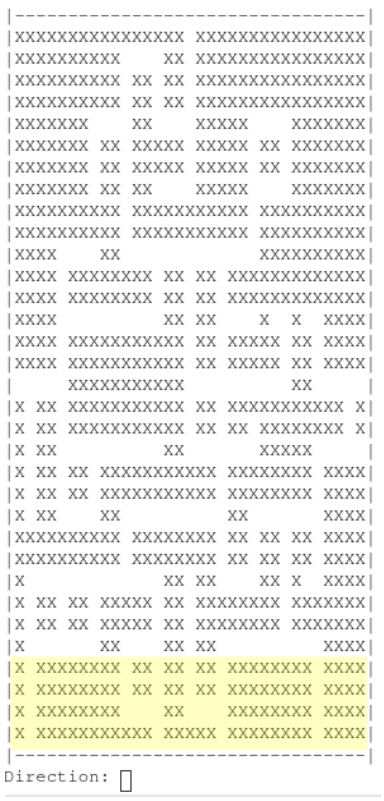
2.2.2 Output

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.

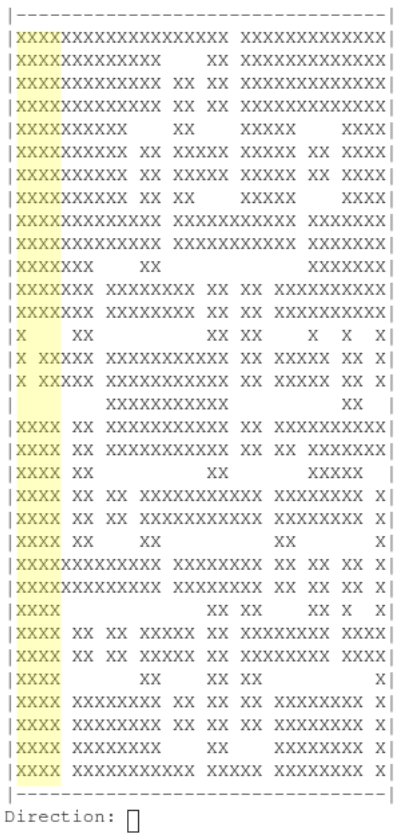
The first output



After moving south (downwards)



After moving east (leftwards)



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.

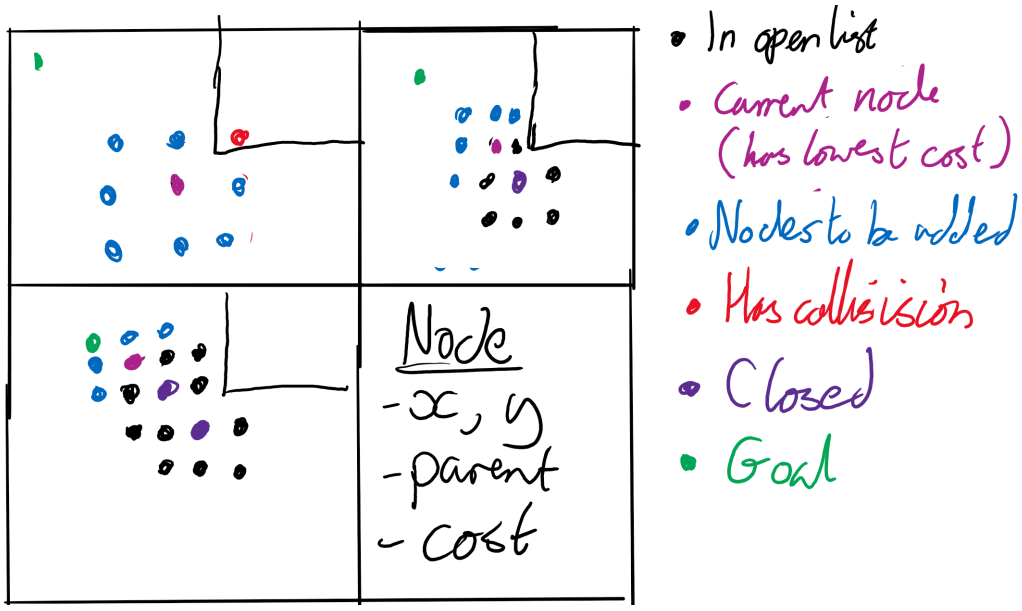


Figure 9: Drawing describing the process of how the A* algorithm finds the shortest route

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).

2.4.2 Characters

2.4.3 Enemies

2.4.4 Rooms

2.5 General Design

2.5.1 Stats

Each stat will influence part of how you play the game. These will be upgraded through finding specific items in chests and experience gained from doing different activities like exploring or attacking.

- 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.
- 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.

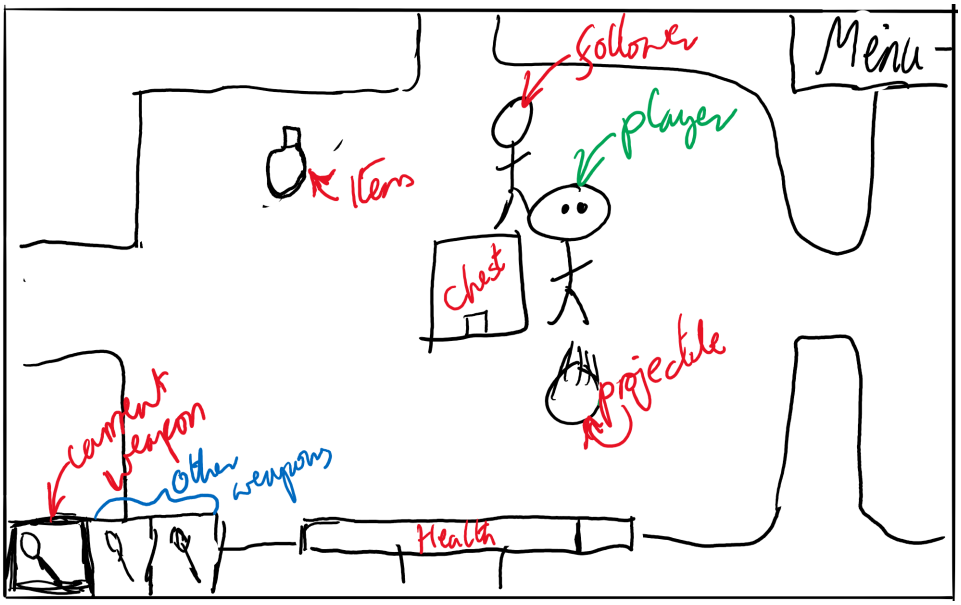


Figure 10: Overall Design plan

- 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
- ShaderEffects - 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

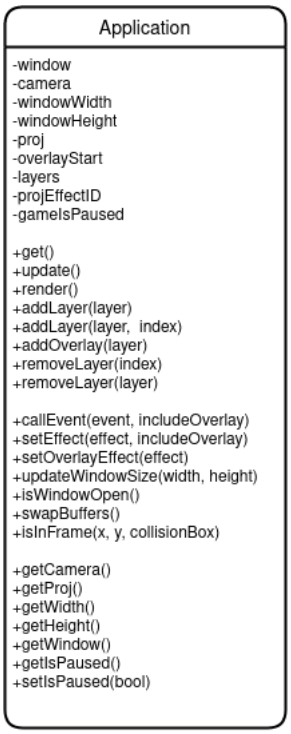


Figure 11: Application class (singleton)

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

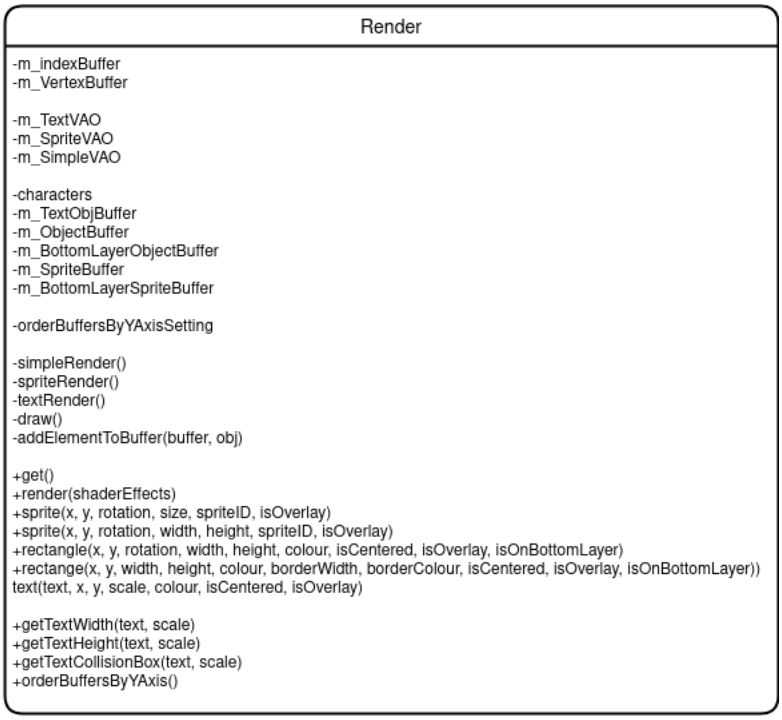


Figure 12: Render class (singleton)

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

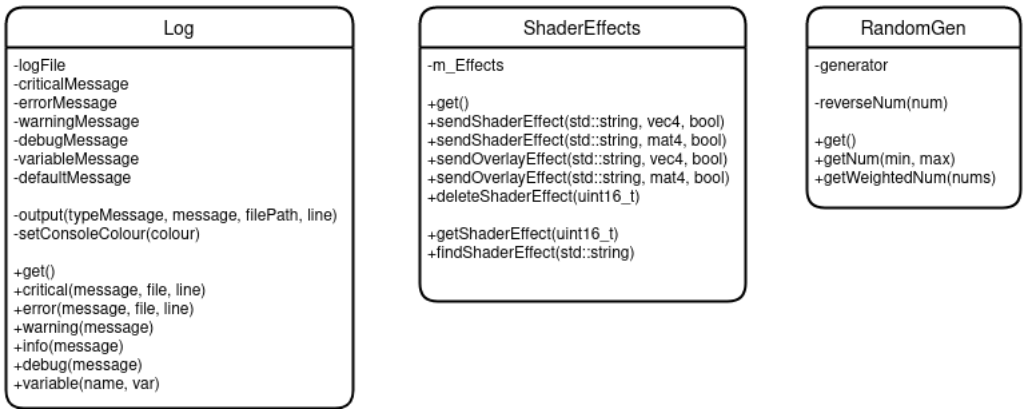


Figure 13: Other singleton classes

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

ShaderEffects

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 ShaderEffects 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

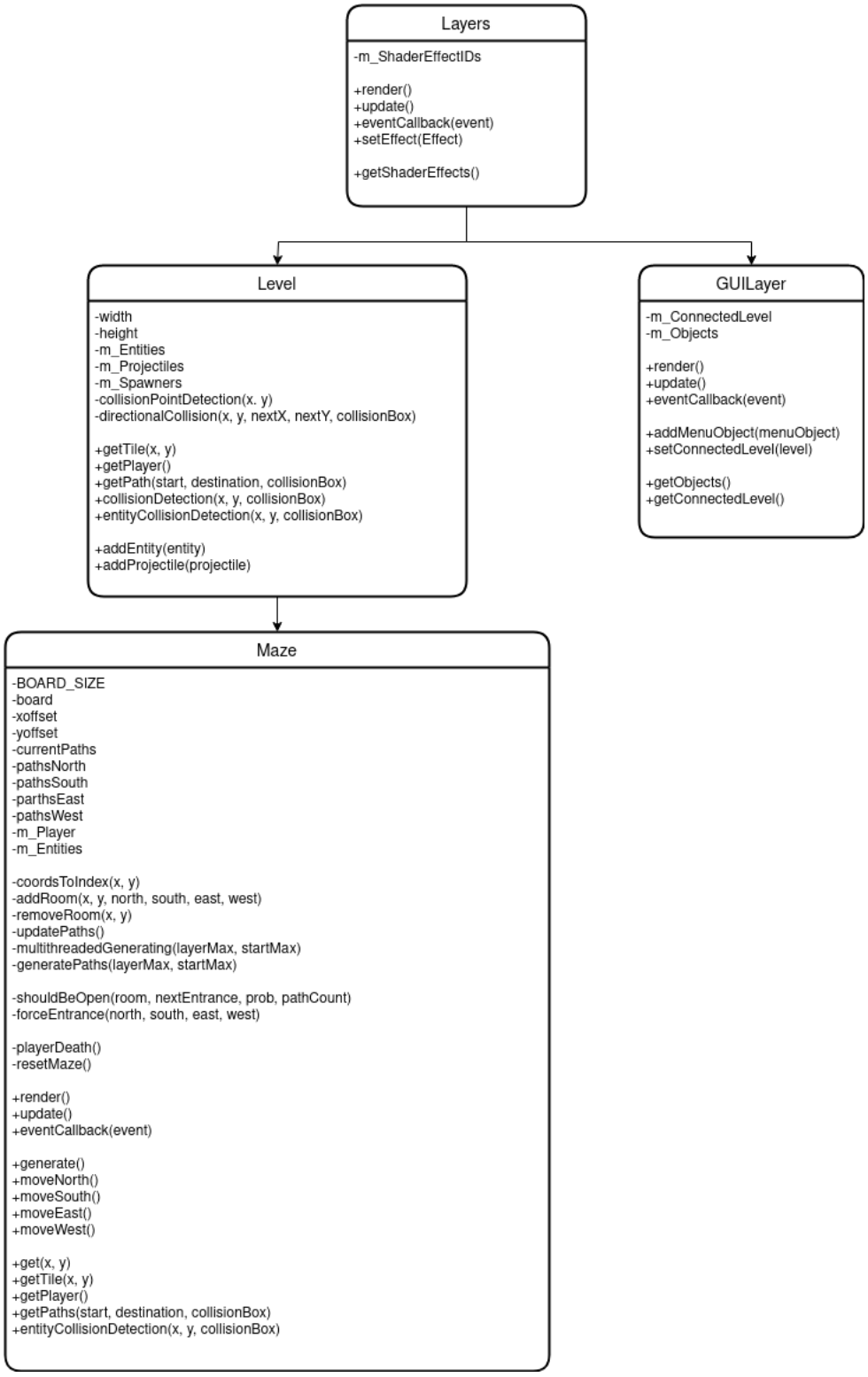


Figure 14: Layer subclasses

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

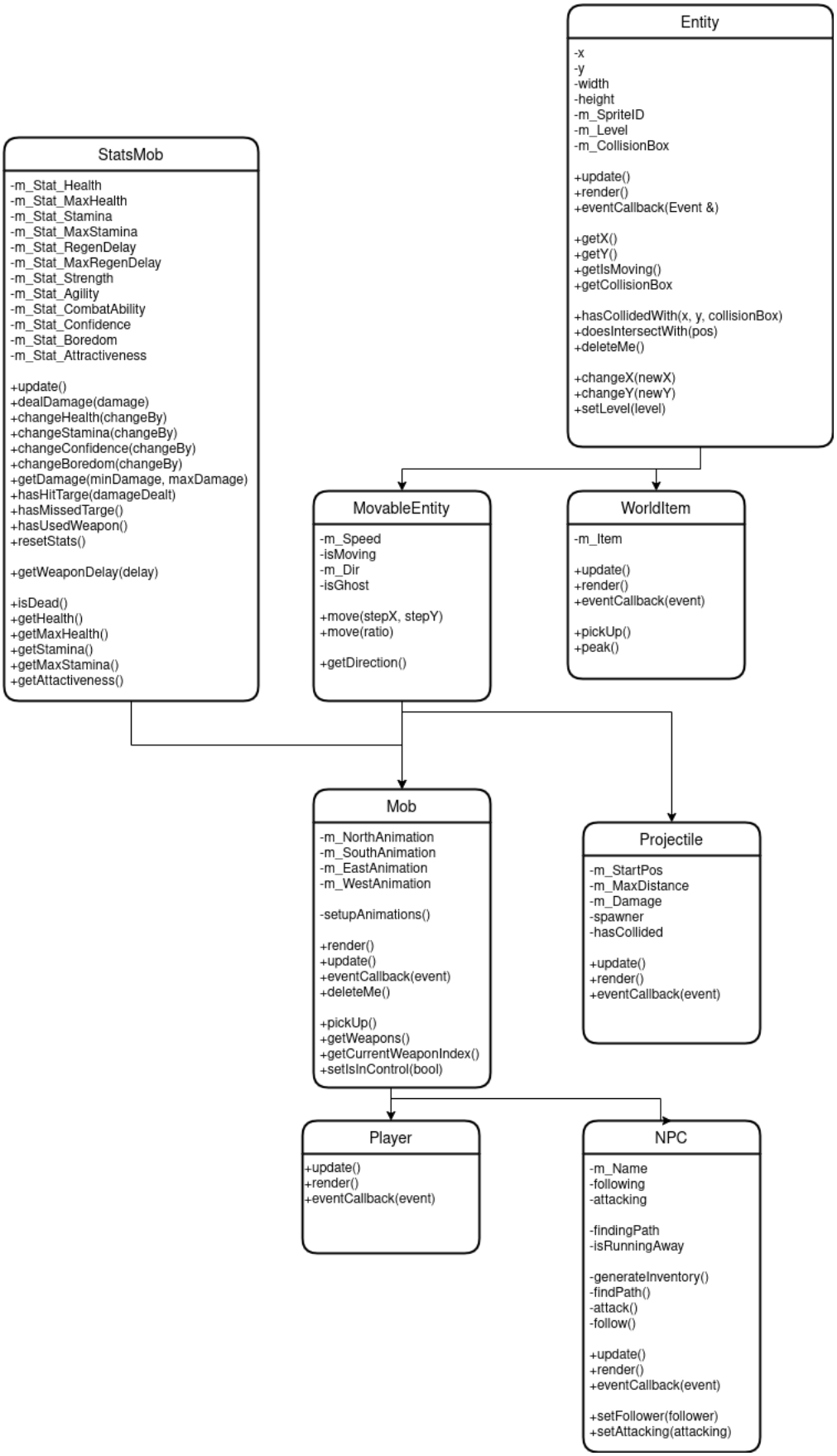


Figure 15: Entity subclasses and StatsMob

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

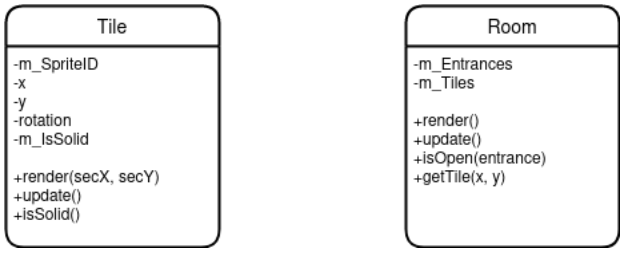


Figure 16: Classes for the design of the maze

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

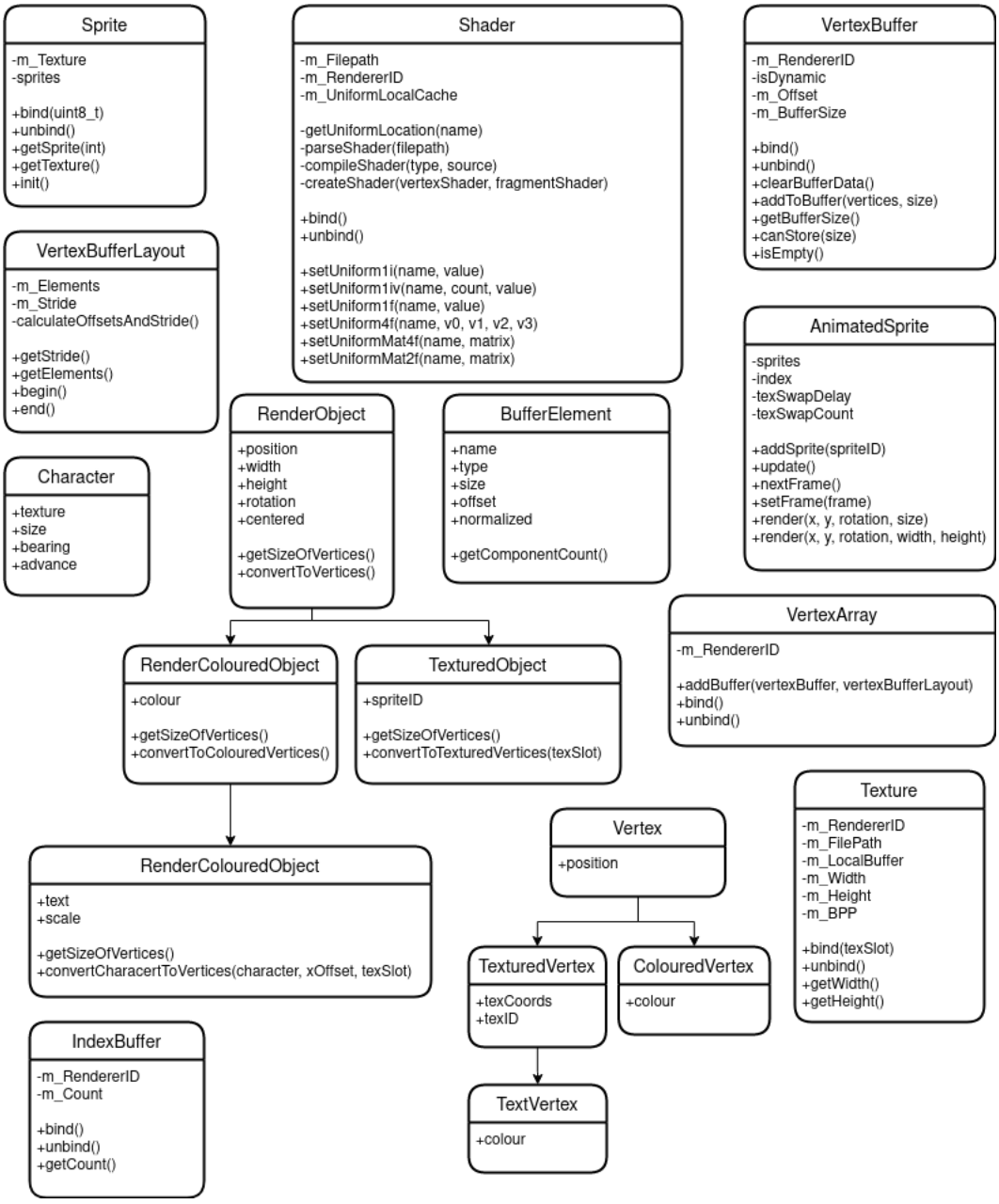


Figure 17: Classes involved in rendering

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

RenderColouredObject

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

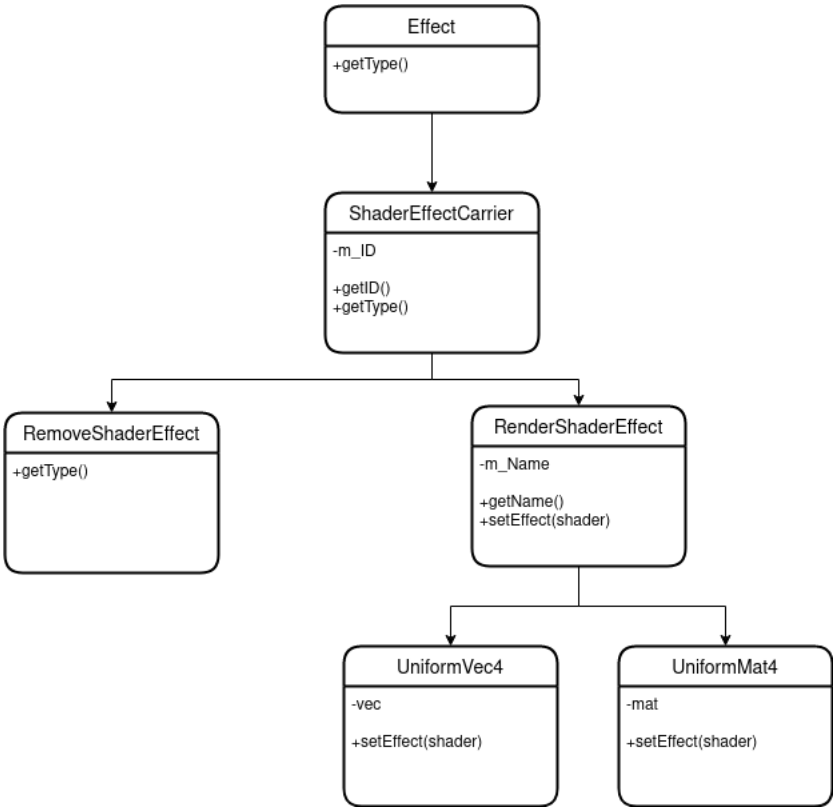


Figure 18: Effect subclasses

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
RenderShaderEffect

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

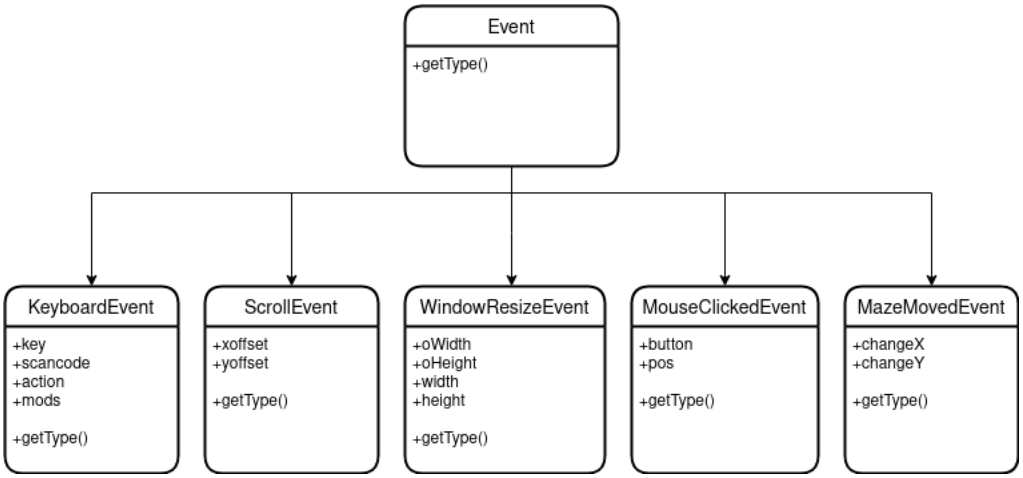


Figure 19: Event subclasses

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

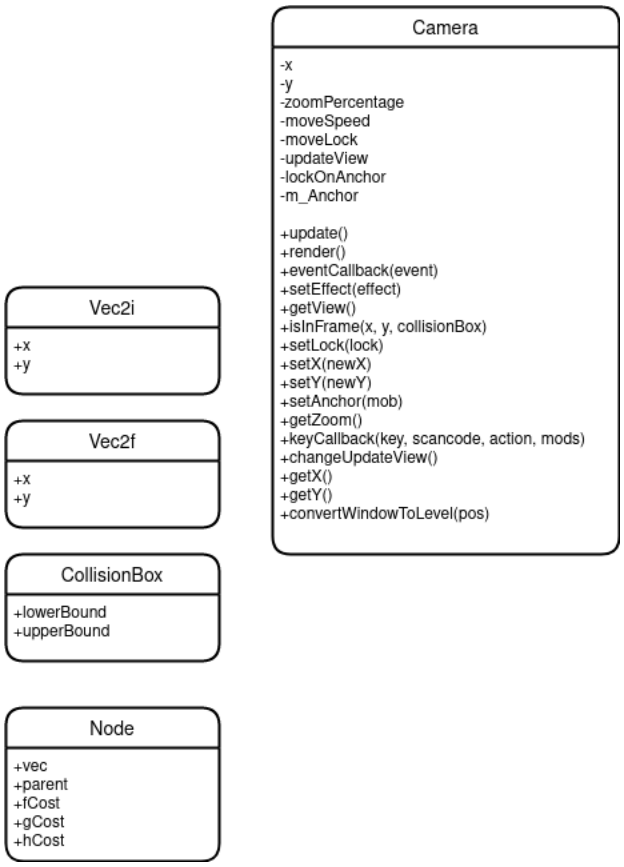


Figure 20: Classes that are for general use

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