NEA - The Maze Game

Hugo Whittome

September 2020

Contents

1 Analysis

1.1 Overview

This is an exploration game, where you explore 5 levels of randomised mazes collecting items and followers on your way to help defeat the monsters.

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?

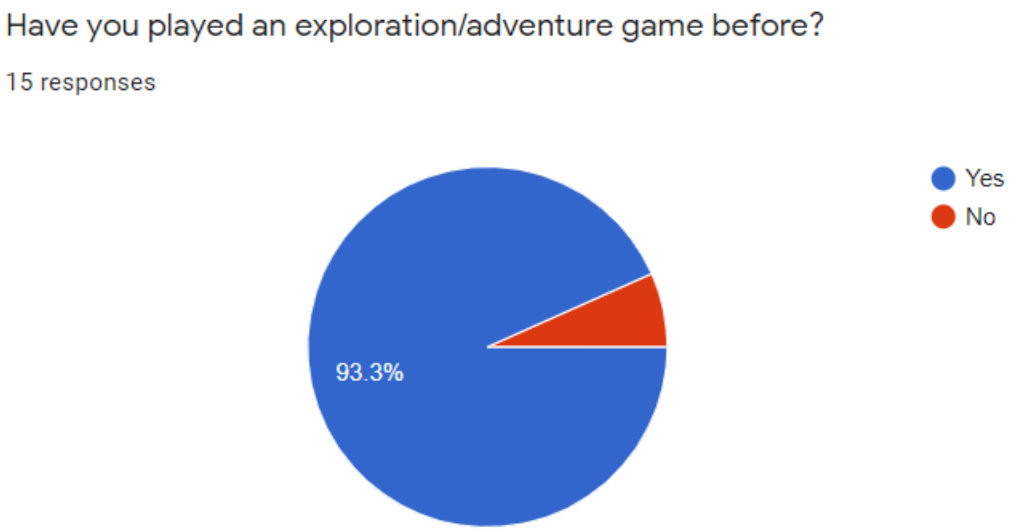


Figure 2: Responses from survey showing most people have played an exploration game

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

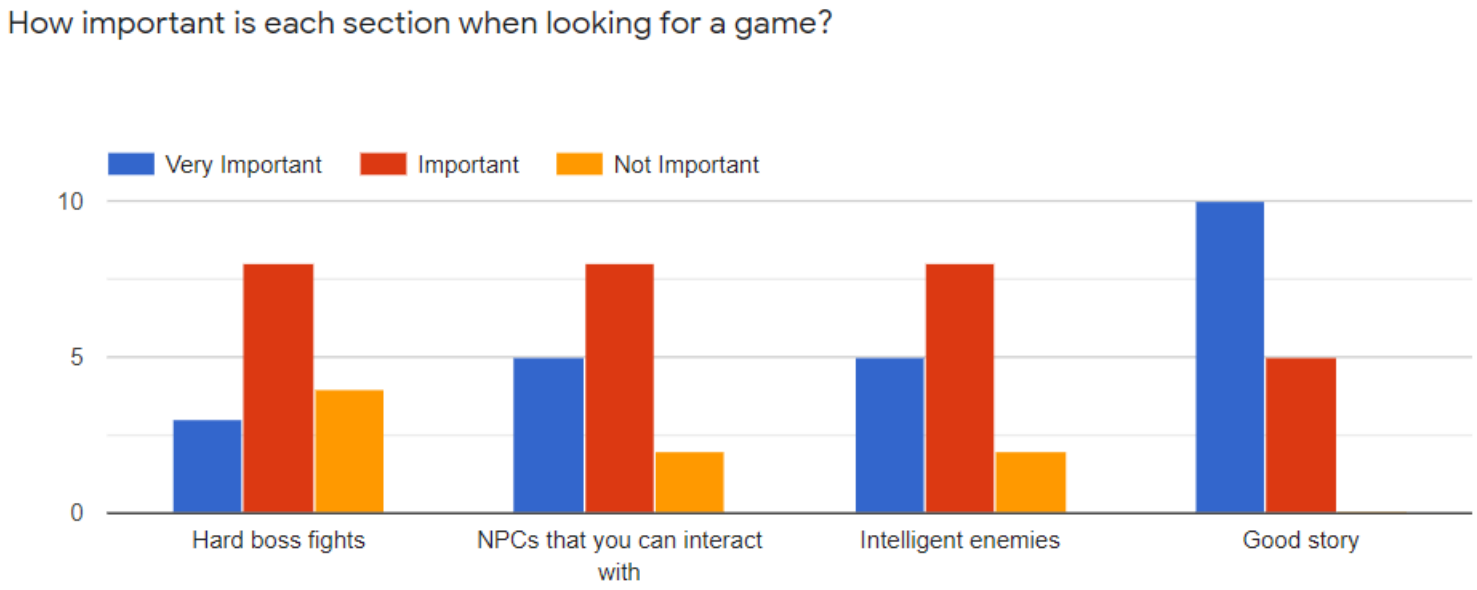


Figure 3: Responses from survey showing that a good story is very important in a game

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

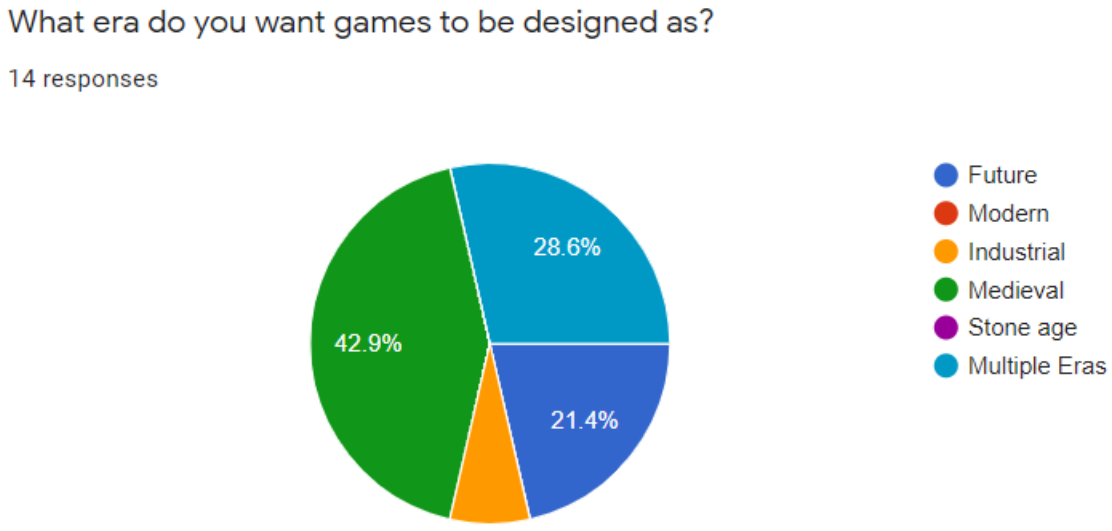


Figure 4: Responses from survey showing that most people like a Medieval design

- Do you like games to have a story based ending or just a location to get to (e.g. flag at the end of Super Mario Bros Levels)?

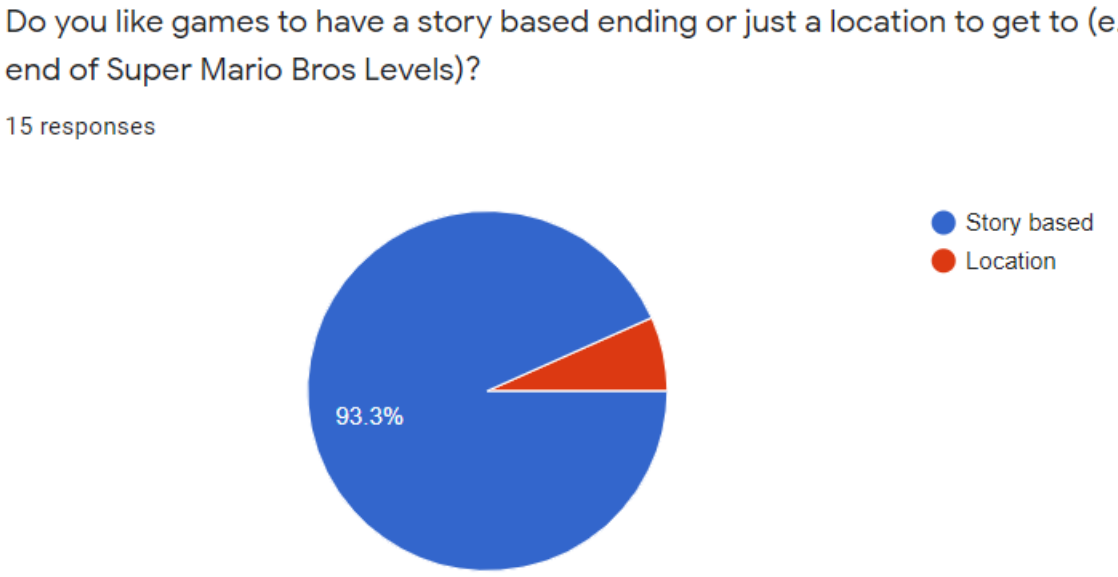


Figure 5: Responses from survey showing the ending should be story driven and not just a point which the player has to get to

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

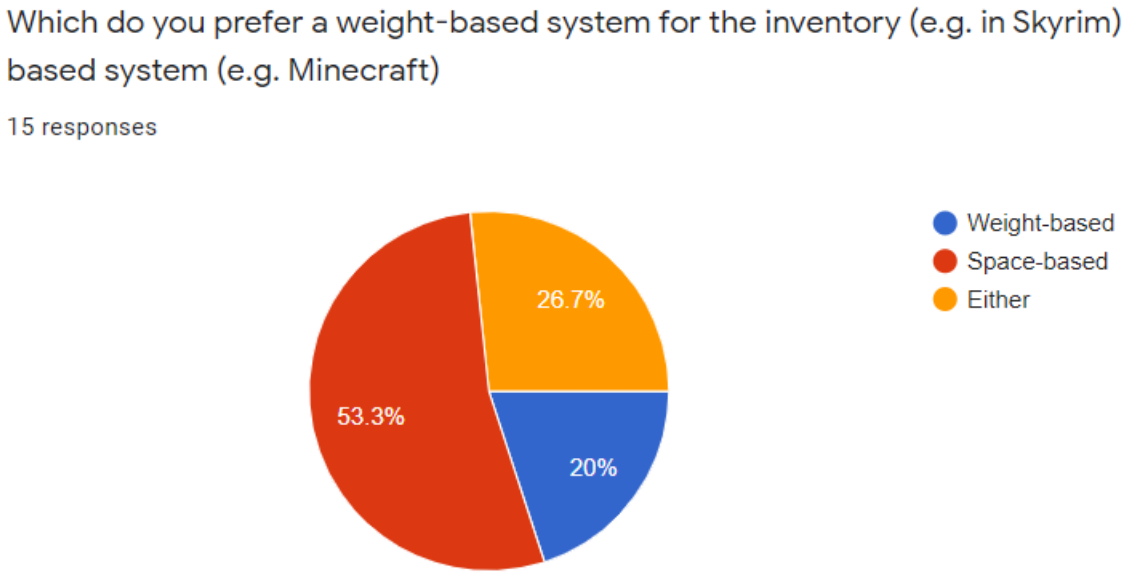


Figure 6: Responses from survey showing that most people like a weight-based system in a game

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

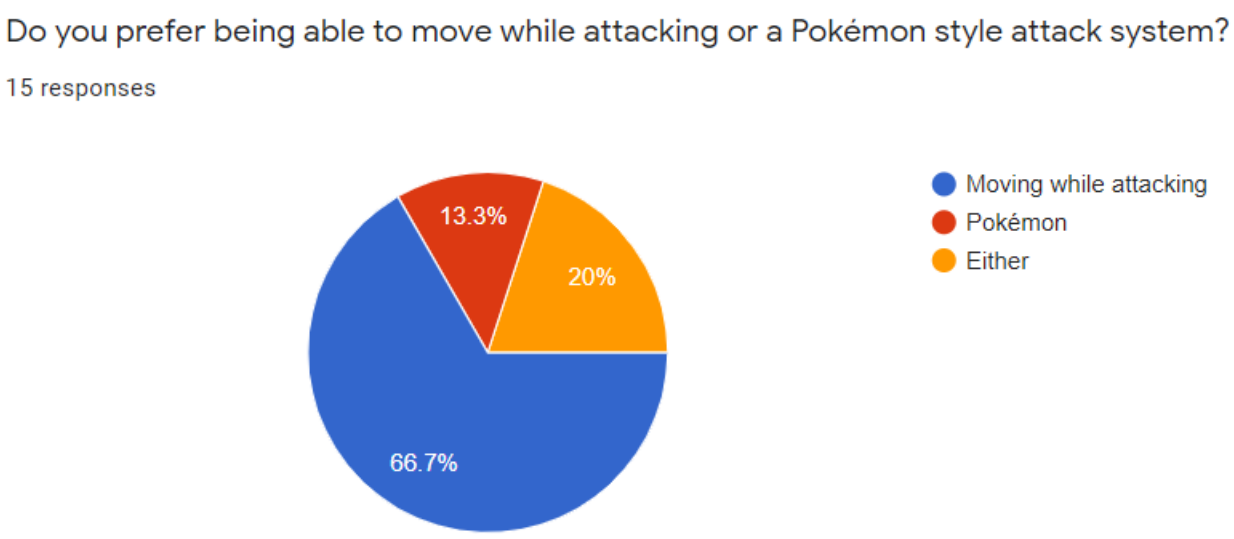


Figure 7: Responses from survey showing that the attacking system should allow you to still control the player

- Have you played "The Binding of Isaac"?

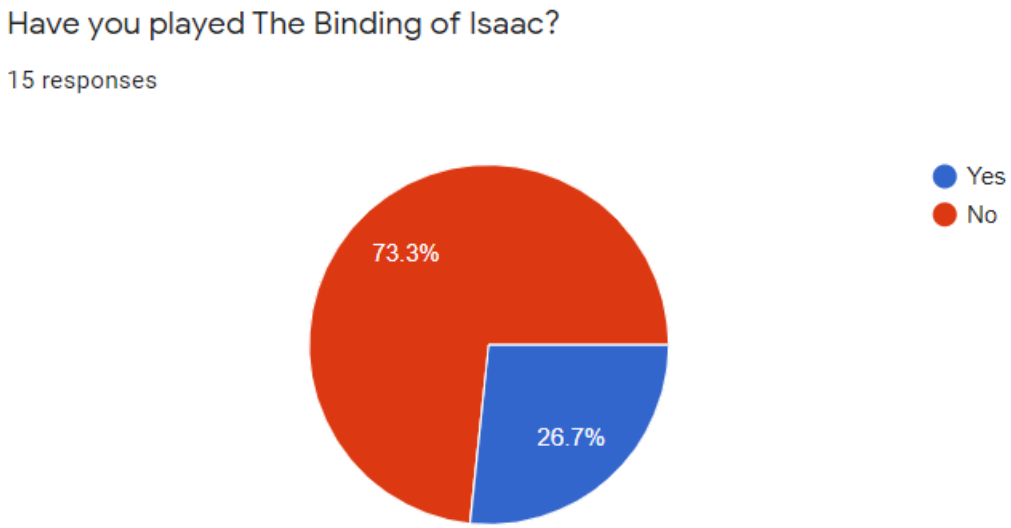


Figure 8: Responses from survey showing that most people in this survey had not played The Binding of Isaac

  - 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

This research has made me prioritise the story as most people believed it to be very important in a game (as shown in Figure 3), this means that I will have to come up with a good story that gives an explanation to why the player is exploring a constantly changing maze. As an extension to this, it seems (as shown in Figure 5) that most people want a story-based ending, meaning that I will have to come up with an ending that suits the game. Furthermore, 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 6, 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 7) people prefer it over an attacking GUI.

As shown in Figure 8, 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 Story

1.5.1 Introduction

As seen from the questionnaire, it is very important for this game to have a good story. So this is the rouge outline of the story that I have planned for this game.

1.5.2 Background Infomation

In this world, there 5 five kingdoms (Ignius, Caelium, Ridiere, Regnio, Labyrinthius) listed below. Each live very differently with different beliefs and have different strengths. These kingdoms have a constant hostility and distrust towards one-another, which has resulted in many battles. The player's character will be the second born of Labyrinthius.

Description of each kingdom:

- ***Ignius*** - A kingdom in the far south of the land. There kingdom is built ontop of the fire mountain, a volcano. This means that most of their buildings are black as they use the stone found around the volcano. Also, they use the lava found inside to build machinery. They use this volcano to through prisoners as a sacrifice to their god, Pozhar. They believe that Pozhar protects them from the wrath of the underworld, that is trying to escape and kill everyone.
- ***Caelium*** - A kingdom in the north of the land. Their capital is at the top of a mountain, looking down on the rest of the world. This has resulted them in seeing the other kingdoms below them. This resulted in them capturing people who wonder too far into their land, and using them as slaves. This has resulted in them using the slaves in wars, making their army extremely powerful. They worship the god who holds up the sky, Aakaash. They believe the god protects them from the sky falling onto them.
- ***Ridiere*** - A kingdom in the east of the land. The land around this kingdom is very flat, allowing them to farm a huge amount of crops. The capital is on a river, which allows transport of their crops to the other kingdoms. For this they worship the river, Flumen. The god who allows their crops to grow and deliver them into the mouths who need them. As the river washes away any bad deed, they believe in rehabilitation for their prisoners.
- ***Regnio*** - A kingdom in the west of the land. Known as being great builders. Their land is surrounded by rocky hills, in which they have built their kingdom on. Their defences are beyond anything any other kingdom has built. They do not believe in a god, instead believe in the power of mankind and they see people who believe in a god petty and weak. Their prisoners are treated with a punishment just of their crime, with time spend in a cage hidden below the ground.
- ***Labyrinthius*** - A kingdom in the center of all other kingdoms. A hub for trading and travellers, where many wizards reside as it's renounced for being a hub of knowledge. Their capital was founded around a whole in the ground, known as "de ore Dei" or the mouth of the god. The god is known as Kurtulmak, a living maze. One of the founders fell into the "de ore Dei" (a couple hundred years ago) and came out alive, talking of the wonders of this living maze and the monsters that lie beneath. They became known as a prophet and the only person to survive and be blessed by Kurtulmak. They have a tradition called "prophetam resurrectionem lesu Christi", where, when the second born of the monarch turns 15, they are chucked into the "de ore Dei", with one weapon of their choosing. If they are able to survive and escape the maze, they are blessed by Kurtulmak are crowned ruler over the kingdom before the first born.

Furthermore, throughout the map there are magical orbs, that once they have been put in staffs, that take hold of the power, allows the user to use the orb's power. These welders are called wizards, normally these wizards go out to try and discover more knowledge about these orbs to unlock their full potential. Then there are knights who own land in each kingdom and can be called by the king to come fight for him. These knights are usually very strong and have experience in battle. These people are usually quite aggressive and so have a bad view point around them, and most peasant try to avoid them to avoid conflicts that could result in their death, which has happened and some knights have been imprisoned for it - in the kingdoms in which this applies (all but Ridiere).

1.5.3 NPCs

Throughout the game, the player will be able to find wondering NPCs, who were prisoners chucked down into the mouth to see if they could escape and be forgiven of all their crimes. The list of NPCs is as follows:

- Max - A wizard from Labyrinthius, who was investigating how the maze moves. He was chosen to be the wizard to venture into the maze to look at it up close.
- John - A pacifist from Ridiere, who was falsely accused of murder, and was chucked into the maze as a trial, if he survives, he will be declared innocent.
- Cleo - A warrior from Ignius, wanting to prove herself as the best of the best.
- Tabitha - Homeless servant from Labyrinthius sent down as a punishment for being caught stealing to survive.
- Hope - A builder from Regnio who was admiring the entrance to the labyrinth when she fell in.
- George - Merchant from Caelium who wanted to show that Caelium was better than every other kingdom, so he decided to jump in the entrance
- Chayton
- Emma
- Simon

1.5.4 Events

The player, the second born of the monarch, Leuok, has grown up knowing they will be chucked into the maze at 14, as part of the prophetam resurrectionem leu Christi tradition. This has caused the King to force them to train as hard as possible in combat and navigation without a map. Leuok has succeeded so far, and is extremely good with a sword and bow. The King, is torn up from this event as Leuok is his favourite child, and wants Leuok to succeed because the first born Ong, does not have the capability to be King. He is lazy and extremely selfish, two qualities that are not good for being King.

After an introduction from a wizard, that is recording these events an unknown amount of years after the event, explaining everything in the previous paragraph, the player will be put into a temple that surrounds the whole. They will be given the choose of a one weapon before they go in. Once they have chosen, they will be told to jump into a hole, if they try to escape, they will be met with guards. The King will be seen, watching with a tear in his eyes, because he believes that Leuok will not survive.

Once the player enters the whole, they will start to explore the maze, where they will fight monsters and maybe make friends will the NPCs that have been chucked in the whole previously. After a while of exploring, they will find a way down to the next level, where it will be taken back to the wizard recording the events, where he will interject saying "This is where Leuok found a way to the next stage, where the monsters were stronger, and fewer survived". This will repeat until the player reaches the 5th level.

Once the player has reached the 5th level, they will be looking for the exit to outside (a set of stairs that leads to a door). Once they reached this door, they will be seen travelling through a corridor which will lead to an exit near the Ridiere kingdom. Then there will be a cut seen of Leuok being crowned ruler. The story will then end with the wizard ending it, as if it was the end of a great book.

1.6 Design Outline

1.6.1 Classes

- Layer - This will allow the subclass to be stored as a layer and be updated and rendered by the application.
- Level - This will store width and height of tiles and implement the A* algorithm for finding the shortest path between two roots.
- Maze - This will store the rooms in a grid and implement the generation of the maze.
- Camera - Generates the view matrix, that will allow the player to move around the map.
- Entity - Stores x and y coordinates of an object in a given level.
- Mob - Subclass of Entity, which implements a move function including collision detection.
- Player - Stores the coordinates of the player.
- Follower - Follows the player around and helps to attack the enemy.
- Enemy - Attacks the player and their followers.
- Item - Base class of all items that can be found throughout the maze.
- Projectile - Is a projectile that travels in a single direction and then deals damage to a mob.
- Room - This stores all the tiles
- Tile - This will store a sprite with coordinates relative to the room.
- Sprite - Stores every sprite that needs to be rendered during the game.

1.6.2 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.
- 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.

1.6.3 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.

1.7 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.
  - Alow them to interact with subtitles, so that they can randomly say quotes, with each NPC with their different set of quotes.
- 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 entity).
  - 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.
- Implement the story
  - Design graphics and write script for opening sequence when you create a new game.
  - Implement a system to render the graphics and text when you create a new game.
  - Do the same for the end sequence.
  - Implement multiple levels into the maze, with the enemies getting harder and change the randomness of the items being created.
  - Add graphics and script for when the player finds stairs to the next level.
  - Do the same for when the player dies
- 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.

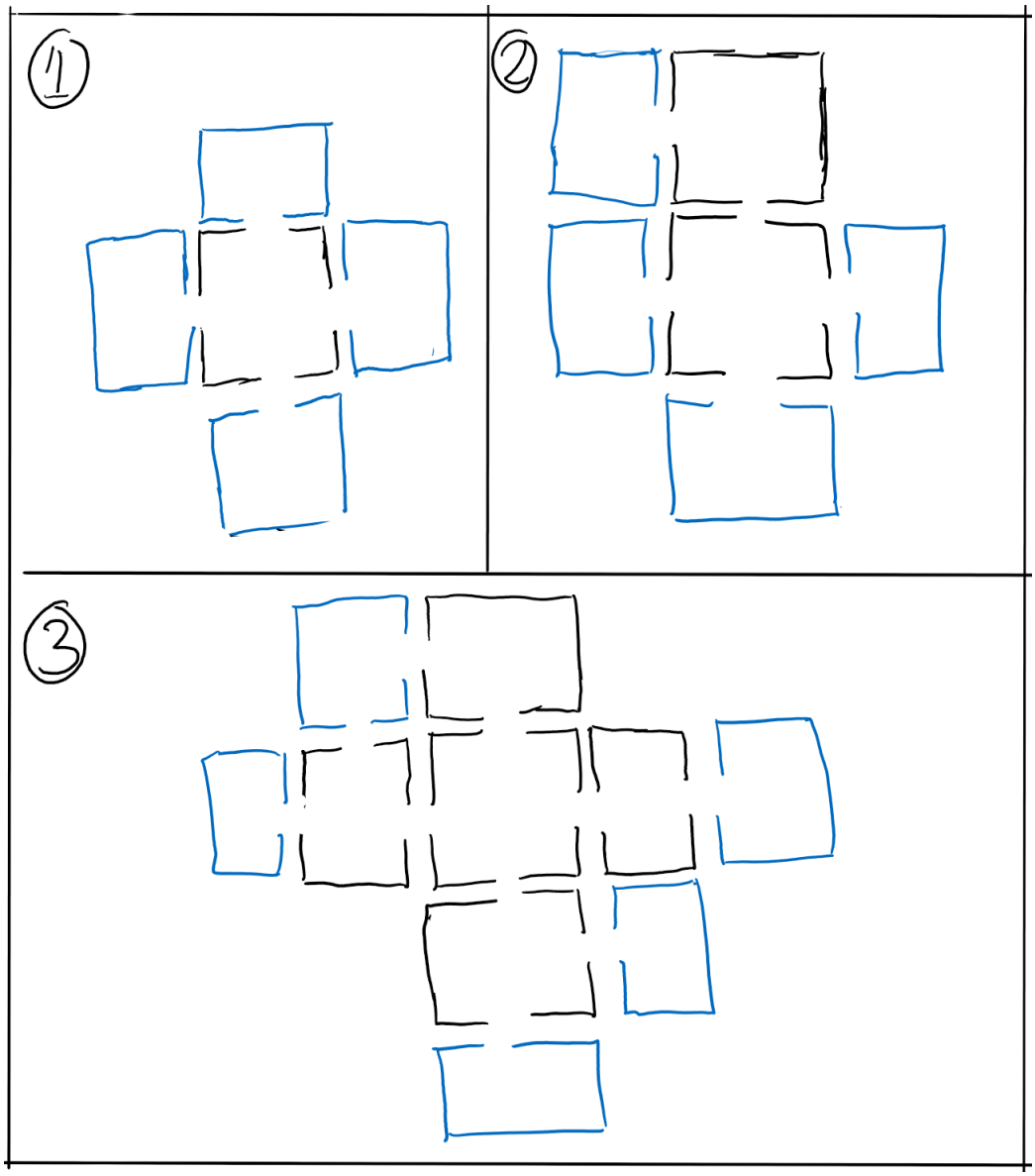


Figure 9: Steps followed by the maze generation

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
```

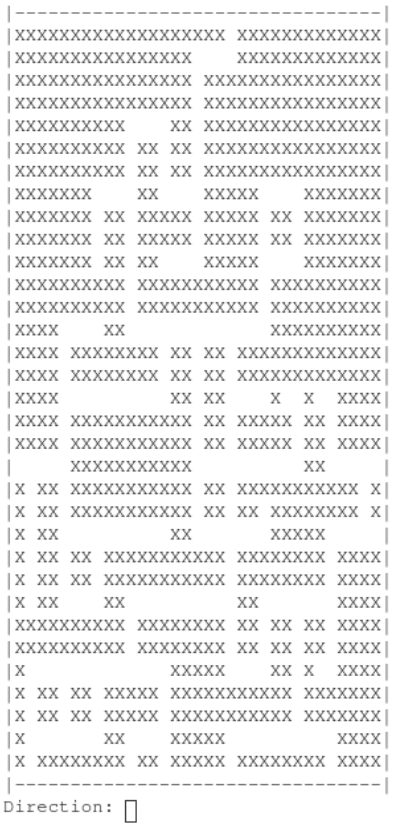
```
                        break
                    c += 1
                r = randint(0, 2)
                if r != 2:
                    r = randint(0, 2)
                    c = 0
                    for i in range(4):
                        if options[i]:
                            continue
                        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 extrem
ely 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 f
unction
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)
```

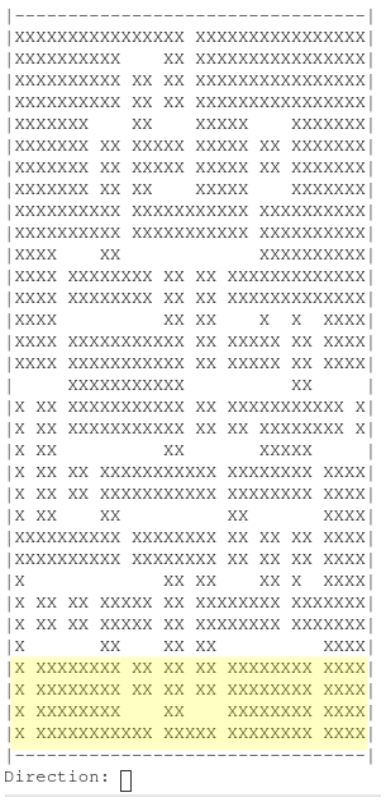
2.2.2 Output

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.

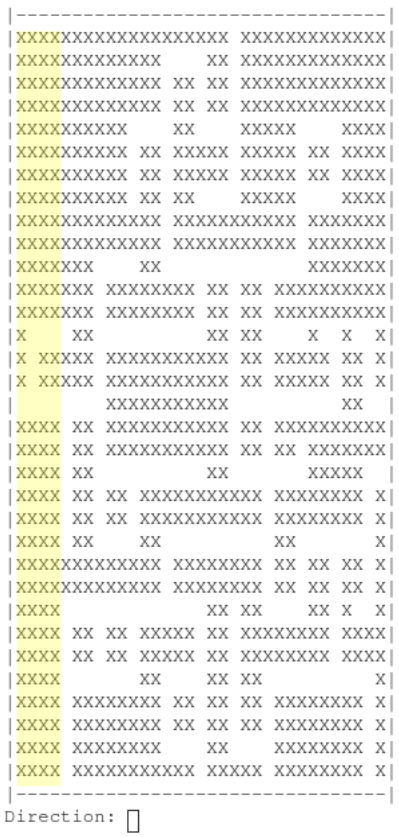
The first output



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.

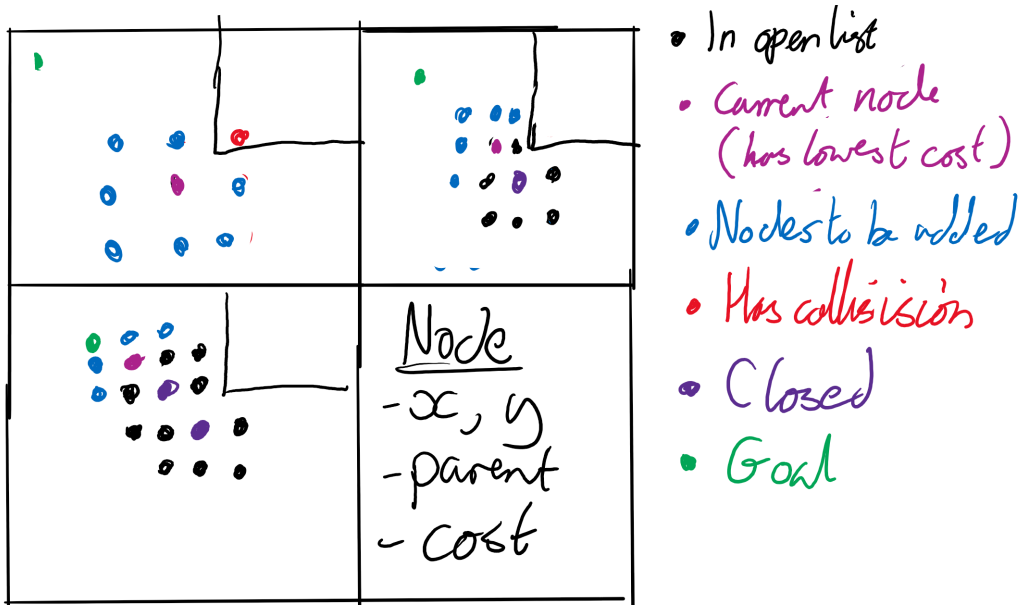


Figure 10: Drawing describing the process of how the A* algorithm finds the shortest route

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.

2.4 Graphical Design

2.4.1 Overall Design

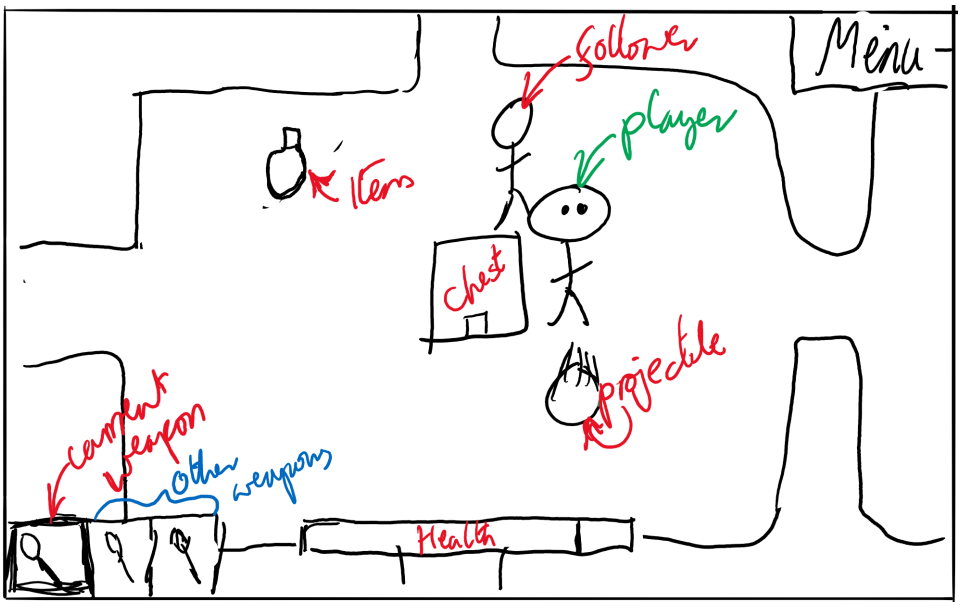


Figure 11: Overall Design plan

2.5 Classes

2.5.1 Layers

2.5.2 Entities

2.5.3 Rooms

2.5.4 Rendering classes

2.5.5 RenderEffects

2.5.6 Other

2.6 Storage

2.6.1 Tiles

2.6.2 Structures

2.7 Functions

2.7.1 Application

2.7.2 Rendering