LOST CAUSE

A-level Computer Science NEA
Chloe Johnson

Contents

Analysis

This is documentation for a website-based rogue-lite game with levels based on a series of procedurally generated mazes. It will be built upon a variety of technologies, including the Flask application framework for Python and Object-Oriented Programming in JavaScript. The application will procedurally generate mazes using a Python backend to be sent to the client.

Background

Why did I choose this project?

I have always been interested in games that offer longevity and replayability without repeating the same activities over and over. This can generally be achieved in two ways, either by continuously updating the game or by using procedural generation. The downside of having to continuously update a game is the large time commitments required by developers over the game's lifetime, to create more new levels and experiences for the end users. Procedural generation however, especially in the example of roguelike games, offers an efficient way of creating a game which is different every time you play, whilst retaining similar challenges and structure for users to play through.

The idea of a maze based game enticed me as it seemed like an interesting and exciting way to experience a roguelike dungeon crawler (a game where the player must make their way through a series of rooms and levels, known as a 'dungeon'), as players would have the added challenge of navigating through the maze, on top of any enemies they might have to fight or challenges they might have to complete.

What is procedural generation?

Procedural generation is a method of producing data automatically, without the need for it to be manually designed. In games, it is when an aspect of the game is controlled by algorithms which determine what that aspect is, either randomly or based on a set of criteria. Examples of procedural generation include:

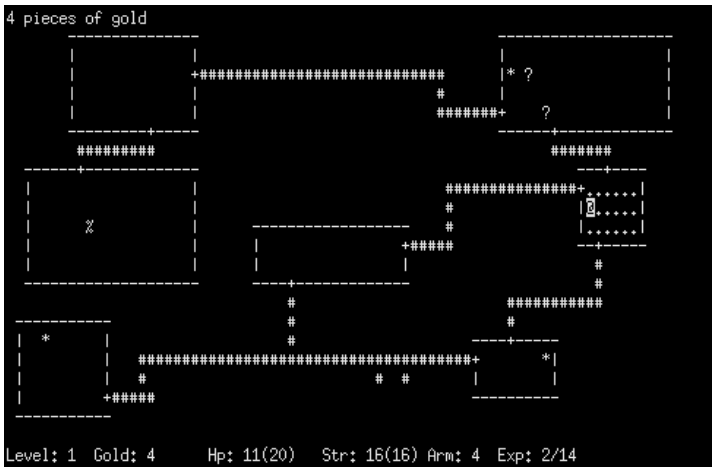
- Changing loot players receive depending on how far through the game they have progressed, e.g. giving more powerful rewards to higher level players
- Generating terrain or a game map, allowing near infinite possibilities of completely different worlds with one piece of code.
- Deciding where, when, how many, and what enemies should be spawned, e.g. based on the area of the map, the in-game time, the difficulty of a level or the level of a player.

Many popular games such as Minecraft, Terraria, Stardew Valley, No Man's Sky, the Civilisation series, the Borderlands series, Rogue, and Rogue Legacy use procedural generation to enhance the player's experience, offering slightly unique experiences each time they are played.

Procedurally Generated Mazes in Games

This project will focus on using procedural generation to create mazes. This style of game is not one with many well-known or renowned games - however, there are still several examples of games that use procedural generation to create mazes. Most of these examples fall into the Roguelike genre.

An in-game screenshot of Rogue



What are Roguelike and Rogue-lite Games?

Roguelike is a genre of games characterised by a dungeon crawl through procedurally generated levels, with permanent death of the player (i.e. you lose all progress on death). 1980's *Rogue*, the namesake of the genre, is commonly considered as the original roguelike - although some games matching the criteria were released before it - and is a terminal based ASCII game, where the player would navigate through a series of dungeons, and everything was represented using different ASCII characters.

Over time, many more roguelike games were released, and the genre grew in popularity. However, some of the more modern games were not incorporating all of the features of a typical roguelike, by allowing players to save data between runs or using procedural generation, for example. Because of this, the term 'rogue-lite' was proposed to make the distinction between these games and true roguelikes.

An Interview with a Prospective End User

This project will be aimed at:

- Users who enjoy the roguelike and rogue-lite genres of games
- Users who regard mazes as interesting and engaging challenges
- Users who appreciate the ease of access of a web-based game

The following is an interview with a prospective end user, someone who fits into all of the above categories.

1. What genres of games do you enjoy playing?
I enjoy a variety of game genres however I prefer roguelikes, puzzle games, and action-adventure games
2. Do you find games with fixed level based structures enjoyable?
Yes I do, I think it can allow developers to create balanced and enjoyable challenges as the game progress
3. Do you think you would enjoy a game based on navigating mazes? If so, why?
I do think I would enjoy a game about navigating mazes. I think that mazes are not only incredibly intellectually stimulating but would also be a great and versatile medium to keep a player engaged
4. Would you prefer more randomness over fixed structure in a game, or a balance of the two, and do you think this would increase the time you spend in the game?
I think some randomness is required in order to keep a game feeling fresh and unique however, too much randomness would make it impossible for a player to be invested in the experience so I think a balance between the two options would be the best way to increase the time I spend in a game
The interviewee desires a balance between randomness and fixed structure
5. Do you think that a good example of the balance would be keeping game mechanics the same throughout, but randomising the layout of the map or terrain?
I think that is an excellent example of balance. Keeping the core game mechanics as a rock while having randomised elements such as layout and enemy spawns can provide a unique and gripping experience every time you play
Consistent game mechanics, but changing the maze for each level is a good example of this balance. They would also expect a game such as this to have some kind of enemy system
6. Looking at these maze examples, which do you think best suits a rogue-lite style game?
The maze that utilises recursive backtracking is the most difficult and would pose the greatest blend of challenge and interesting gameplay while rewarding the player for significant problem solving. The other maps are poor due to the simple pathways, multitude of short dead ends, and lack of real player choice as choosing the wrong path will result in immediate failure and backtracking which would be disheartening.
The interviewee regards the recursive backtracking algorithm as the best in terms of the shape of the maze, and how it will have to be solved
7. Would you like to have a level of control over how each maze generates? If so, what factors would you like to be able to change or control?
I think that it could significantly improve the game after completing the main game to have access to a type of “free play” that would allow them to create mazes based on their own parameters in order to increase the longevity of the game.
They would like have the option to generate custom mazes once they have completed the rest of the game

8. Is there anything else you would like to see from this project? Are there any other requirements that you have?
I would like to be able to play the game from multiple devices and have my save data transfer between these devices. Furthermore, it would be very pleasant to be able to play the game on the go or when I am out and about. I would also like to see another way of completing each maze than simply finding an exit, for example, completing some kind of puzzle, as this would allow the game to stay interesting as you progress through the levels
The interviewee would like to be able to play the game on any device, and have their progress save between those devices. They would also like for mazes to be completed by completing tasks, such as puzzles throughout the maze.

As evidenced by their answers to the first few questions, the interviewee is a good example of a prospective user of the game, as they prefer roguelikes and puzzle games, and enjoy solving mazes and playing games that involve them.

Summarising the results of the interview, it is clear that the project should include:

- Recursive-backtracking as the main algorithm used to generate mazes
- Game levels that feature randomly generated mazes, but keep core mechanics the same throughout
- 'Enemies' to face within each maze
- A 'free-play' mode after a user has completed the levels, which would allow them to have more control over how the mazes are generated
- Easy portability of the game
- A way to complete levels other than finding an exit

Modelling the Project

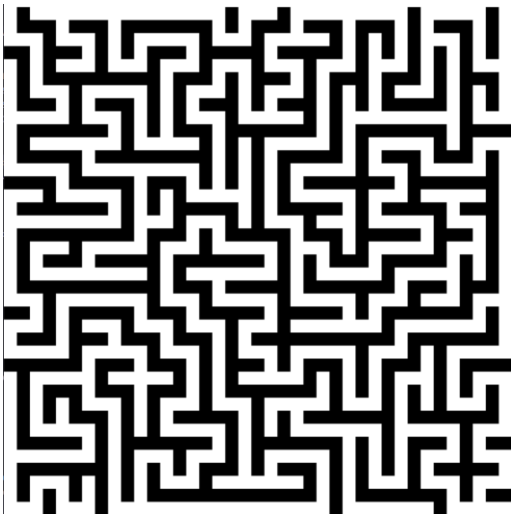
Sources:

http://www.jamisbuck.org/presentations/rubyconf2011/index.html#title-page
http://weblog.jamisbuck.org/2011/1/3/maze-generation-kruskal-s-algorithm

Image source:
https://en.wikipedia.org/wiki/Maze_generation_algorithm

An analysis of Maze Generation Algorithms

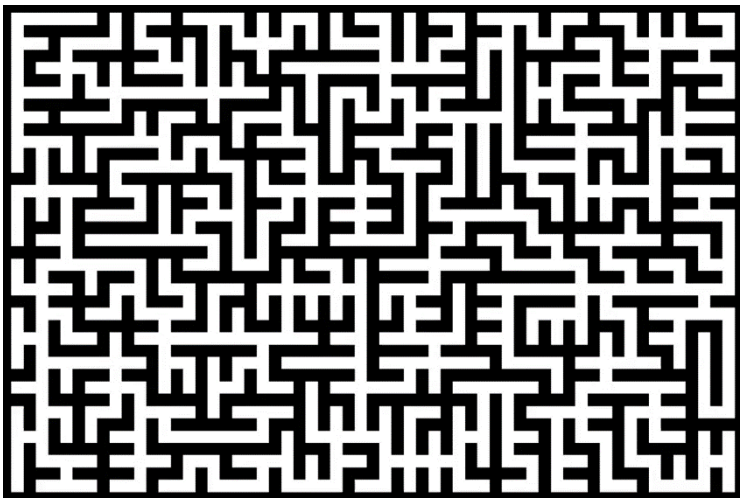
Recursive Backtracking



Recursive backtracking is an algorithm that is used for both generating and solving mazes. The algorithm does a random 'walk' until it reaches a dead end, then will check each previously visited node until it can continue again. This algorithm produces mazes that look like typical mazes - having long winding corridors and not too many dead ends.

The time complexity of the recursive backtracking algorithm is O(2n) where n is the number of nodes in the maze. This is because each node will be visit once as the algorithm randomly walks around the maze, then again when it backtracks through.

Kruskal's

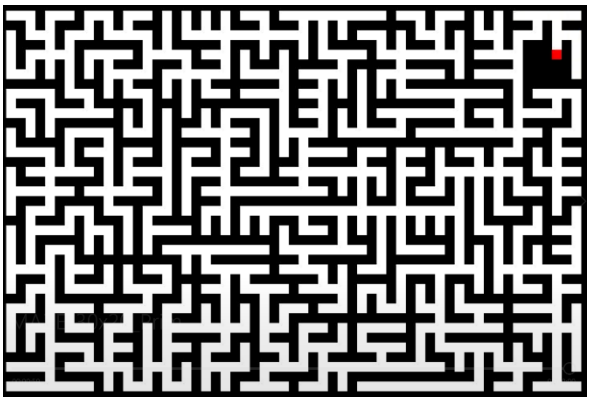


Kruskal's algorithm is one that was designed to produce a minimum spanning tree from a list of weighted nodes. By giving each node a random weight, it can be used to produce a maze. Each node starts as its own 'group' (known as a disjoint set) then two adjacent nodes are chosen to be merged into one group, bringing any other nodes in their group with them. The maze is complete when all the nodes are in one group. Unfortunately, Kruskal's algorithm produces mazes with lots of very short dead ends, in an almost repetitive pattern, which, as confirmed by the results of the interview, is not as well suited to a maze as the recursive backtracking algorithm.

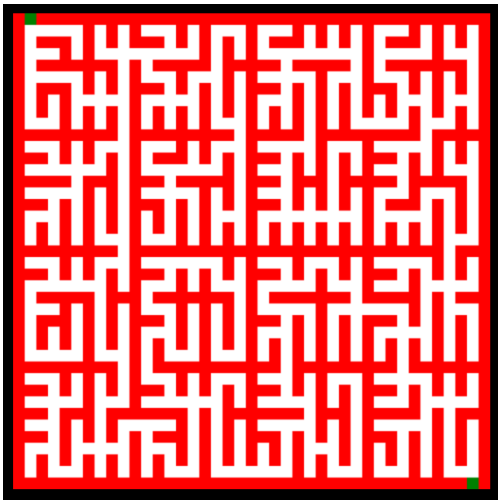
The time complexity of Kruskal's is O(nlogn), where n is the area, due to the implementation of a disjoint tree.

Primm's

Primm's algorithm, like Kruskal's, is also for producing a minimum weighted spanning tree. From the starting node, each adjacent node is added to a list, then a random node is chosen to be visited, until there are no nodes in the list. Unsurprisingly, as Primm's is a similar algorithm to Kruskal's it produces fairly similar mazes. However, the mazes are worse than those generated by Kruskal's as they end up with several 'branches' off from the starting node, that if the player takes, they will have to return to the start to choose a different branch - almost as if they were exploring a tree, not a maze.



The time complexity of Primm's is O(n), where n is the area, as each node is always visited once.



Recursive Division

Recursive division is a fairly different algorithm to the others. It divides the area in two at a random point, then adds a path at one position in the wall. It will repeat this process for each of the sub areas created until there are none left. The main problem with recursive division is that the mazes are very predictable and simple to solve. As you can see, this maze has clear divisions between 16 areas, and the mazes becomes 'which major area do I visit next'.

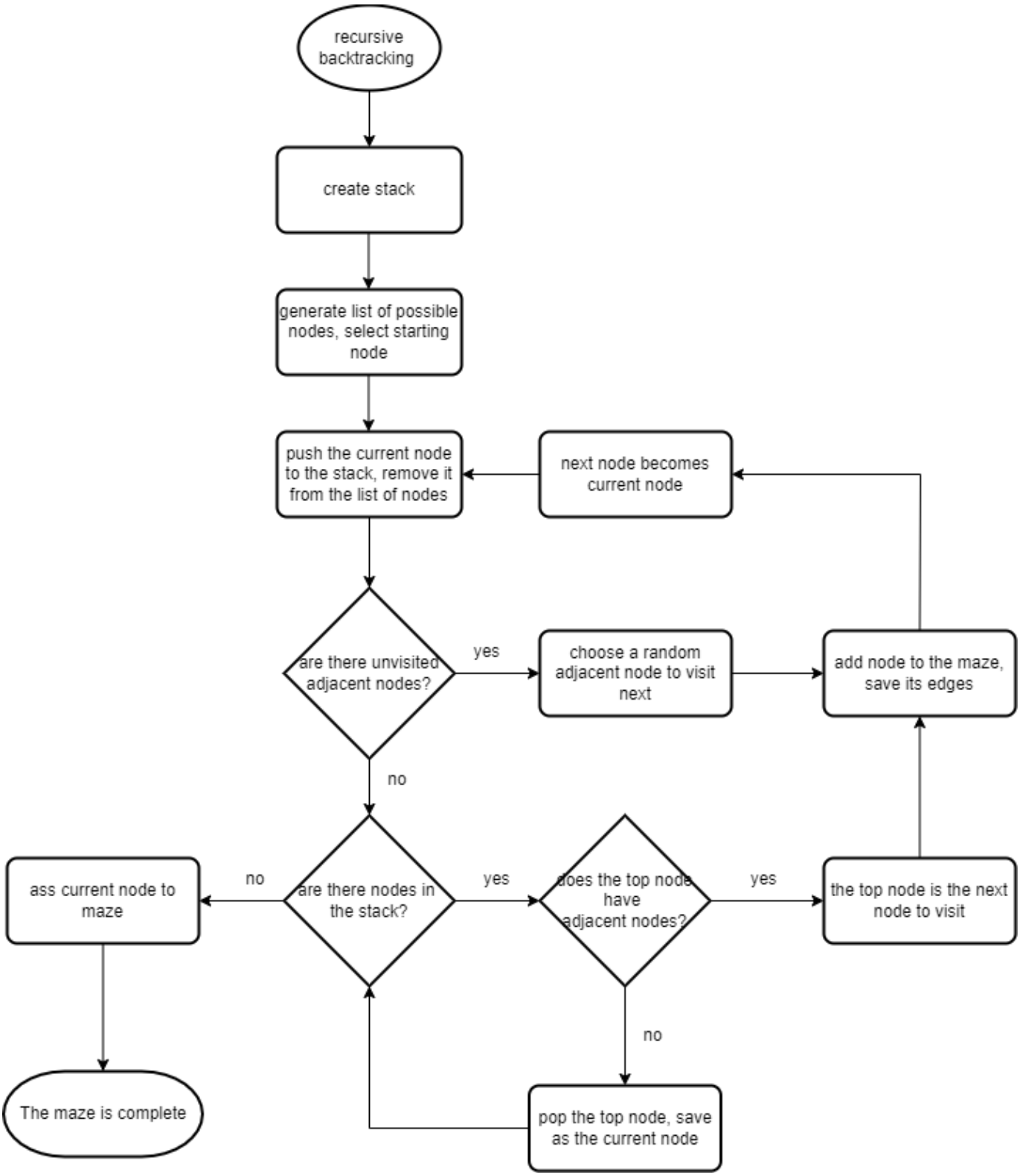
The time complexity of the recursive division algorithm is O(nlogn).

Why is Recursive Backtracking the Most Suitable Algorithm?

When using a recursive backtracking algorithm, it is relatively easy to adapt the program to allow large variety in how mazes generate. For example, by choosing a node randomly from the stack (a list in this instance) the maze will generate differently. There is also the possibility of adding weights to the adjacent nodes chosen to visit, causing a tendency for more horizontal or vertical corridors.

Due to the nature of the other algorithms described, none of them allow this level of control over the generation of the maze. However, Kruskal's algorithm can allow 'weaver' mazes to be created, if node groups are allowed to merge under - or over - other nodes.

Ideally, the efficiency of the algorithm would be better - however, in the context that it is being used, the mazes won't be extremely large, so can still be generated in short times.


recursive backtracking
create stack
generate list of possible nodes, select starting node
push the current node to the stack, remove it from the list of nodes
next node becomes current node
are there unvisited adjacent nodes?
yes
choose a random adjacent node to visit next
add node to the maze, save its edges
no
ass current node to maze
no
are there nodes in the stack?
yes
does the top node have adjacent nodes?
yes
the top node is the next node to visit
no
The maze is complete
pop the top node, save as the current node

Analysis of a Similar Existing Solution

Sources:

https://retrospiritgames.blogspot.com/2014/07/instant-dungeon-roguelike-maze-game-for.html
https://store.steampowered.com/app/326720/Instant_Dungeon/

Instant Dungeon! is a roguelike maze game, and is similar to the idea of this project

From its page on Retro Spirit Games:

Instant Dungeon is a cute little maze game available on both PC and the PlayStation Vita (via the PlayStation mobile section on the store) which features roguelike elements to keep things interesting. Created by lone developer Scott Mattot, *Instant Dungeon* keeps things incredibly simple by tasking you with little more than moving around a succession of randomly generated, top down mazes, avoiding the skeletons, vampires, zombies and other monstrosities, nabbing any treasures along the way and finding the key to open the exit door. As you progress deeper into the dungeons the light get more scarce, meaning danger could lurk around every corner.

Instant Dungeon has several game modes, however, to unlock most of them, you have to play the standard “Adventure Mix” mode, which is a series of fixed levels, without randomly generated mazes, many times. There is a mode available with random mazes, but they do not increase in difficulty without leaving the game and selecting a harder difficulty. This would be a problem for prospective users of this project as having to complete the same repetitive levels over and over, with no random generation is not particularly enjoyable, and goes against what the interviewee suggested.

On top of this, the portability of Instant Dungeon is not particularly feasible, as data on the steam version does not transfer between devices, something that is also required by users of this project.

However, there are many strengths to Instant Dungeon. Aside from the mazes, the structure of each level and the consistent game mechanics are places that this project can definitely take valuable inspiration from. Essentially, each level requires finding a key to unlock a door (which also has to be found), allowing progression to the next level. Also, defeating more monsters increases your score for the level and the overall playthrough. This method of having multiple simple objectives is an effective way of introducing and increasing how challenging the game is. Throughout the mazes, players can obtain weapons and armour/shields to help them fight the monsters, this adds novelty to each level, as players become stronger as the environment becomes more challenging. Once they finish a playthrough, Instant dungeon contains a feature where users can share their high scores with other users and view other users' scores.

Objectives based on Instant Dungeon review:

- An objective, such as a key or several pieces of keys for the player to complete, and a lock mechanism to use it on.
- Some form of hostile “enemies” for the player to face
- Items, such as armour or weapons, that the player can use to help them

- A scoring system

Overall objectives

1. The application should generate random, procedurally generated mazes.
    1.1. Using the recursive backtracking algorithm
        1.1.1. The mazes should have variable size
            1.1.1.1. Side lengths of width and height can be either different or the same
            1.1.1.2. Upper limit for side length of 25 squares
            1.1.1.3. Lower limit for side length of 3 squares
    1.2. The mazes should be saved as a bitmap image
        1.2.1. These images should be available to the webpage
    1.3. Mazes should be shareable between different users
2. The application should be hosted as a website
    2.1. The application should be available on any HTML 5 supporting browser
    2.2. The application will not be available on touchscreen devices *(see 4.1.4.1)*
3. There should be a series of 10 levels of progressive difficulty for the user to play through
    3.1. Each level should be a random maze
        3.1.1. Each maze should have a fixed size
        3.1.2. The maze size should increase with the levels
    3.2. There should be hostile non-player characters (enemies) in the maze
        3.2.1. More enemies should spawn as the size of the maze increases
        3.2.2. The enemies should find a path to and follow the player
        3.2.3. The enemies should attack the player, reducing the player's health by 1
        3.2.4. The enemies should have a health bar to display how much health they have remaining
    3.3. Each level should contain items
        3.3.1. Items can be picked up
        3.3.2. Items can be carried
        3.3.3. Items can be dropped in a new location
    3.4. There should be a player character in each level
        3.4.1. This should be controlled by the user using a keyboard
        3.4.2. The character should not be able to leave the space the maze inhabits
        3.4.3. The character should have a certain amount of health
            3.4.3.1. The amount of health the character has should be proportional to the size of the maze
            3.4.3.2. The character should have a health bar to display how much health they have remaining
            3.4.3.3. If the character's health drops to 0, the game should end and the user should be returned to the menu
                3.4.3.3.1. The user should have to restart from the first level
            3.4.3.4. The character should have an inventory to carry up to 5 items in
            3.4.3.5. The character should carry a weapon item that allows them to attack enemies

- 3.5. Each maze should have objectives that the user has to complete to finish the level
  - 3.5.1. The objectives should be a series of locks and keys placed around the maze
    - 3.5.1.1. The player should be able to pick up keys to use on locks
    - 3.5.1.2. The locks should be in fixed positions
    - 3.5.1.3. When a key is used to open a lock, that lock should be removed
  - 3.5.2. There should be more objectives to complete as the size of the mazes increases
  - 3.5.3. Once the user has completed the objectives for a level, they can either:
    - 3.5.3.1. Try the level again
    - 3.5.3.2. Move onto the next level
- 4. Once all 10 levels are completed, the user should unlock a “Free Play” mode
  - 4.1. Maze parameters should be able to be specified for a custom maze, such as:
    - 4.1.1. Maze height and width
    - 4.1.2. Maximum number of enemies
    - 4.1.3. Maximum number of objectives
  - 4.2. The user should be able to re-try any of the mazes they have completed previously
- 5. There should be a scoring system for the levels
  - 5.1. The score should be based on:
    - 5.1.1. Time spent on the level
    - 5.1.2. Number of enemies defeated
    - 5.1.3. Number of objectives completed
    - 5.1.4. Size of the maze
  - 5.2. The score should be displayed to the user after the level

Objectives to user requirement mapping and roguelike game-design

User Requirement/ Aspect of Roguelike genre	**Objective**
Recursive-backtracking as the main algorithm used to generate mazes	1.1
Game levels that feature randomly generated mazes, but keep core mechanics the same throughout	1, 4
Enemies’ to face within each maze	3.1.2
A ‘free-play’ mode after a user has completed the levels, which would allow them to have more control over how the mazes are generated	4
Easy portability of the game	2
A way to complete levels other than finding an exit	3.1.5
Procedurally generated levels	1
Loss of all progress on character death	3.1.4.2.2
Next level automatically unlocked when level is completed	3.1.5.3

Acceptable Limitations of the Project

The project will have support for mobile or touch screen devices (e.g. phones and tablets) as this would require special website formatting and support for touch screen controls - both of which are out of the scope of this project.

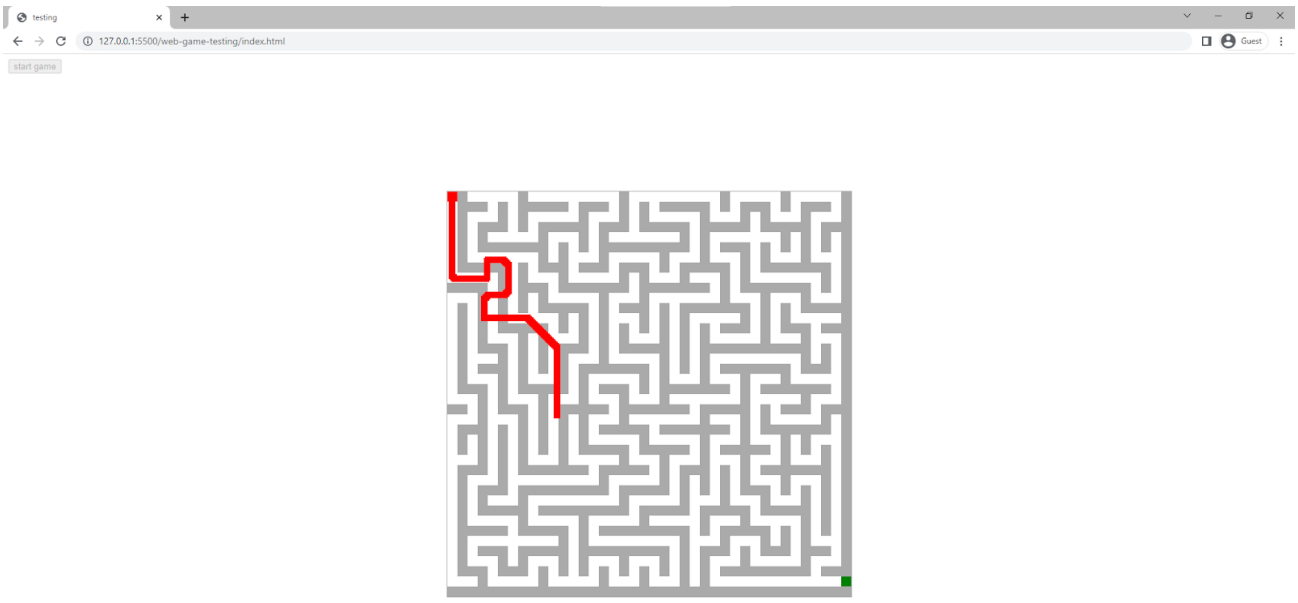
As the game will run predominantly on the client side (in a web browser) the program should not have to account for a user using the editor feature provided by most modern browsers to make changes to the game's function.

Prototyping the Solution

Knowing that the project will be web-based and will use a recursive backtracking algorithm to generate the maze, I chose to make a prototype for the application.

First, I created a recursive backtracking algorithm in Python, which, based on a height and width for a maze, will generate a "spanning tree" - a list containing each node in the maze in the order that it was visited, and the node that was visited before it.

The program saves this data to a file as text, which is then read by the front-end JavaScript, which draws the maze on an HTML canvas object. Using WASD, you can draw on top of the maze, but as you can see, there is no collision detection.



As I have never used JavaScript to this extent before, I followed an online tutorial (https://www.w3schools.com/graphics/game_intro.asp) so I could familiarise myself with how to control and move an object on the canvas. Following the tutorial, I was originally going to have the user move around a square, not draw a line - but this would require wiping and re-drawing the canvas every frame, including the entire maze - the demo ran at around 5 fps, and was fairly unplayable.

Because of this issue, I decided it would be best to create an image of the maze in the backend, and have that be available to the website so it could be displayed to the user. This also meant that I wouldn't use canvas objects for the actual implementation, so I did not use any of the JavaScript from this tutorial in it.

Prototype Code

JavaScript from tutorial

```
var myGamePiece;
var myObstacles = [];
var myMaze = [];
var spanningTree = [];
myMaze.push(new component(15, 15, "red", 0, 0))
myMaze.push(new component(15, 15, "green", 585, 570))

const xhttp = new XMLHttpRequest();

xhttp.onload = function(){
    spanningTree = (xhttp.responseText).split("\r\n")
    for (i = 0; i < spanningTree.length; i++){
        spanningTree[i] = spanningTree[i].split(",")
    }
    for (i = 0; i < spanningTree.length; i++){
        for (n = 0; n<4; n++){
            spanningTree[i][n] = parseInt(spanningTree[i][n])
        }
    }
}
xhttp.open("GET", "spanning-tree.txt");
xhttp.send()



function startGame() {
    myGameArea.start();
    myGamePiece = new component(10, 10, "red", 2, 2);
    var x, y;
    for (i = 0; i < myObstacles.length; i ++) {
      if (myGamePiece.crashWith(myObstacles[i])) {
        myGameArea.stop();
        return;
      }
    }

    // obstacles
    x = 50;
    y = 0;
    //myObstacles.push(new component(50, 50, "black", x, y));
    for (i = 0; i < myObstacles.length; i ++) {
        myObstacles[i].update();
    }

    // maze
    for (i = 0; i < spanningTree.length; i++){
        coordSet = spanningTree[i];
        x = (coordSet[0]-1)*30;
        y = (coordSet[1]-1)*30;
        myMaze.push(new component(15, 15, "white", x, y));
        x = ((coordSet[2] - coordSet[0]) / 2 + coordSet[0])-1;
        y = ((coordSet[3] - coordSet[1]) / 2 + coordSet[1])-1;
        myMaze.push(new component(15, 15, "white", x*30, y*30));
    }
    for (m =0; m < myMaze.length; m ++){
        myMaze[m].update();
    }
}

```

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 600;
        this.canvas.height = 600;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas, document.body.childNodes[0]);
        this.frameNo = 0;
        this.interval = setInterval(updateGameArea, 20);
        window.addEventListener('keydown', function (event) {
            myGameArea.keys = (myGameArea.keys || []);
            myGameArea.keys[event.keyCode] = true;
        })
        window.addEventListener('keyup', function (event) {
            myGameArea.keys[event.keyCode] = (event.type == "keydown");
        })
    },
    stop : function(){
        clearInterval(this.interval)
        console.log("stopped")
    }
}

function everyinterval(n){
    if ((myGameArea.frameNo/n)%1 == 0) {return true;}
    return false;
}

function component(width, height, color, x, y) {
    this.gamearea = myGameArea;
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
    this.newPos = function() {
        this.x += this.speedX;
        this.y += this.speedY;
    }
    this.crashWith = function(otherobj) {
        var myleft = this.x;
        var myright = this.x + (this.width);
        var mytop = this.y;
        var mybottom = this.y + (this.height);
        var otherleft = otherobj.x;
        var otherright = otherobj.x + (otherobj.width);
        var othertop = otherobj.y;
        var otherbottom = otherobj.y + (otherobj.height);
        var crash = true;
        if ((mybottom < othertop) ||
        (mytop > otherbottom) ||
        (myright < otherleft) ||
        (myleft > otherright)) {
          crash = false;
        }
        return crash;
    }
}
```

```
126  function updateGameArea() {
127      myGameArea.frameNo ++;
128      myGamePiece.speedX = 0;
129      myGamePiece.speedY = 0;
130      if (myGameArea.keys && myGameArea.keys[87] &&
131          !(myGamePiece.y <= 0)) {myGamePiece.speedY = -2; } // w
132      if (myGameArea.keys && myGameArea.keys[65] &&
133          !(myGamePiece.x <= 0)) {myGamePiece.speedX = -2; } // a
134      if (myGameArea.keys && myGameArea.keys[83] &&
135          !(myGamePiece.y >= myGameArea.canvas.height - 10)) {myGamePiece.speedY = 2; } // s
136      if (myGameArea.keys && myGameArea.keys[68] &&
137          !(myGamePiece.x >= myGameArea.canvas.height - 10)) {myGamePiece.speedX = 2; } // d
138      myGamePiece.newPos();
139      myGamePiece.update();
140      if (myGamePiece.x >= 585 && myGamePiece.y >= 570 && myGamePiece.y <= 585){
141          element = document.getElementById("wellDone");
142          element.style.visibility = 'visible';
143      } else {
144          element = document.getElementById("wellDone");
145          element.style.visibility = 'hidden'
146      }
147  }
```

Translating key presses into piece movement

By following this tutorial, the most useful feature I learnt how to implement was a system for character movement, using event listeners to detect key presses, then translating those into movement of an object.

```
import random
import time

# generating nodes based on chosen size
maxX = 20
maxY = 20

nodes = []

for x in range(1, maxX+1):
    for y in range(1, maxY+1):
        nodes.append((x, y))

adjacent_nodes = ((-1, 0), (1, 0), (0, 1), (0, -1))

stack = []
spanning_tree = []

start_time = time.time()

next_node = (1, 1)

while True:

    current_node = next_node
    stack.append(current_node)

    try:
        nodes.remove(current_node)
    except ValueError:
        pass

    possible_nodes = []

    # finding possible next nodes by comparing each position adjacent to the current node to the
    unused nodes
    for dx, dy in adjacent_nodes:
        if (current_node[0] + dx, current_node[1] + dy) in nodes:
            possible_nodes.append((current_node[0] + dx, current_node[1] + dy))
    if possible_nodes:
        # choosing a random (adjacent) node to go next
        next_node = random.choice(possible_nodes)
        spanning_tree.append((next_node, current_node))
    else:
        # checking each node from the stack for possible nodes, if there are none, removing it
        for index in range(len(stack)-1, -1, -1):
            check_node = stack[index]
            for dx, dy in adjacent_nodes:
                if (check_node[0] + dx, check_node[1] + dy) in nodes:
                    next_node = check_node
                    break
            else:
                try:
                    # using stack to keep track of which nodes to/ not to visit again
                    stack.remove(check_node)
                except ValueError:
                    pass
                continue
            break

    if len(stack) == 0:
        break

end_time = time.time()-start_time
print((str(end_time)[:-(len(str(end_time).split('.')[1])-2)]) + 's')

file = open('web-game-testing/spanning-tree.txt', 'a')
file.truncate(0)  # empties the file before writing new tree
for item in spanning_tree:
    #           {      coordinate going to        } {       coordinate coming from    }
    file.write(
        f'{str(item[0][0])},{str(item[0][1])},{str(item[1][0])},{str(item[1][1])}\n')
```

This is the recursive backtracking algorithm I created to generate a maze. I tried a few different methods of performing the necessary logic for the algorithm to properly function, and this seemed to be the most efficient and concise way of implementing it for the prototype. However, some changes will need to be made to fit the requirements set out by the objectives.

The most useful feature I figured out how to implement was the stack – the algorithm uses a list to push nodes it visits to, then pops them when it checks to see if they have any available neighbours.

This algorithm produces a list of coordinates stored in pairs, the first being the current position, and the second being the previously visited position. These values are then saved to a text file in the following format:

Each line is a coordinate pair, in the same order as described above.

(for a 20x20 maze, the file is 400 lines long)

```
1   1,2,1,1
2   1,3,1,2
3   1,4,1,3
4   1,5,1,4
5   2,5,1,5
6   3,5,2,5
7   3,6,3,5
8   3,7,3,6
9   3,8,3,7
10  4,8,3,8
11  4,9,4,8
12  4,10,4,9
13  4,11,4,10
14  5,11,4,11
15  5,12,5,11
16  5,13,5,12
17  5,14,5,13
18  6,14,5,14
19  6,13,6,14
20  6,12,6,13
```

Design

High Level Overview

Programming languages environments and libraries

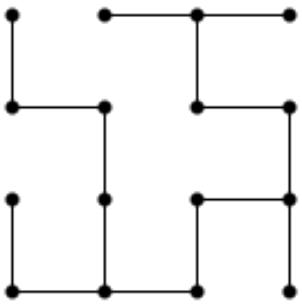
This project will include:

- A Python based backend
- Use of the Flask library for Python to manage communication between the server and client
- Use of the Jinja library for easier management of HTML templates
- A frontend built with HTML, CSS, and JavaScript
- Use of the Bootstrap library to enhance HTML and CSS

Languages and External Libraries Utilised:

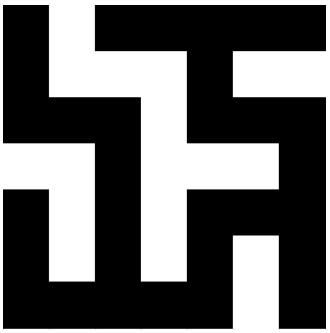
Python	Used to build the backend of the site
Javascript	Used to program the front end of the website and control the game
HTML	Used to structure the webpages
CSS	Used to structure the webpages
Flask	Used to host the server and communicate with the client
Pillow (PIL)	Used to create the maze images
Jinja	Used to allow for dynamic HTML templates
Bootstrap	Used to complement CSS

Key terms and defining a maze



The definition of a maze that has been adopted by this document is that of what is technically referred to as a perfect maze – a maze in which there is only ever one path between any two places in the maze. This is very similar to that of a spanning tree, in fact, a maze of height and width 4 can be described as a graph with 16 nodes, arranged in 4 rows and 4 columns, where the path of the maze is determined by a random spanning tree of the graph, for example:

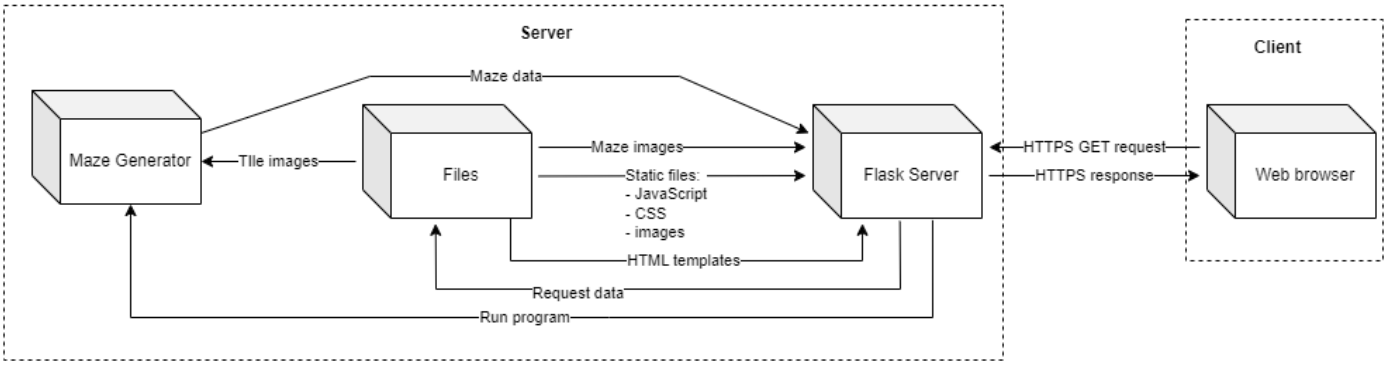
If, from this layout of nodes and edges, we were to draw a series of corridors, it would look something like this:



As you can see, the nodes can have edges, or corridors, adjacent to them in any direction, but the edges have either a node above and below them (a vertical edge) or nodes to the left and right (a horizontal edge). Therefore it is only necessary to store the properties of the nodes of a maze to know its full layout. This is the distinction between what are referred to as nodes and edges throughout this document – however, as in the completed maze, there is no obvious visual distinction between nodes and edges, the term “tile” will be used to refer to both nodes and edges.

The Client-Server Model

This project follows a client-server model typical of a Flask webapp. The diagram below gives an overview of how the different aspects of the server interact and communicate with each other. The communication between the server and client is fairly simple, as the Flask app only processes HTTP GET requests, and returns the appropriate file/s and/or template to the client.



File structure description

The following folders and images describe the various files and components of the system, and where they are located within the app.

File name	Path	Purpose
requirements.txt	/	Contains required Flask dependencies
run.py	/	Runs the app
.gitignore	/app	Non-essential to the program, tells git what folders and file to exclude from commits

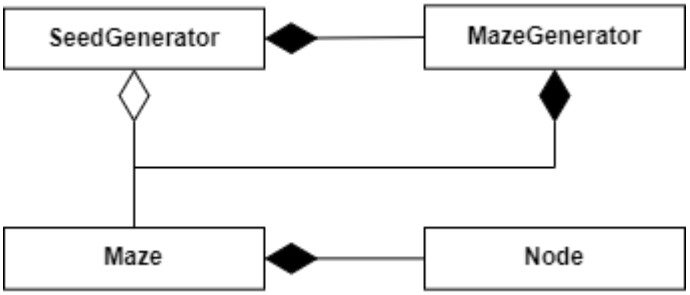
__init___.py	/app	Initialises the app
views.py	/app	Main code for flask app, contains function for each web app location
generate_maze.py	/app	Contains code to generate random mazes and seeds
style.css	/app/static/css	CSS stylesheet for general web pages
level.css	/app/static/css	CSS stylesheet for levels
bootstrap.min.css	/app/static/css	Bootstrap stylesheet
app.js	/app/static/js	Program to load and run the maze levels
bootstrap.bundle.min.js	/app/static/js	Bootstrap javascript
index.html	/app/templates/public	HTML for home page
play.html	/app/templates/public	HTML for levels
gamecomplete.html	/app/templates/public	HTML for the game complete page
public_template.html	/app/templates/public/templates	Jinja2 template for public web pages
level_template.html	/app/templates/public/templates	Jinja2 template for level web pages
active_slot.png	/app/static/img	Image displayed to indicate currently active inventory slot
cursor.png	/app/static/img	Custom cursor image
gamecomplete.png	/app/static/img	Message displayed on the /gamecomplete page
gameover.png	/app/static/img	Message displayed if user loses a level
inventory-bg.png	/app/static/img	Displayed as background for inventory, shows each individual slot
key.png	/app/static/img	Sprite for the key item
levelcomplete.png	/app/static/img	Message displayed when user completes a level
lock.png	/app/static/img	Sprite for the lock item
lostcausetitle.png	/app/static/img	Title image for the home page
player-spritesheet.png	/app/static/img	Sprite Sheet for the player character
slime-spritesheet.png	/app/static/img	Sprite sheet for the enemies
sword.png	/app/static/img	Sprite for the sword (weapon) item
*.png	/app/static/img/maze	Maze tiles to build the maze image with
*.png	/mazeimages	Completed maze images, to be served to the client

The Backend

generate_maze.py

generate_maze.py UML Diagram

The UML diagram for generate_maze.py, showing the association between classes



Class Descriptions for generate_maze.py

Format:

: protected, + : public, - :private

Class *ClassName* extends *Parent*
Description

Properties:
#/+/- property_name: data type <<has get or set methods>>
Description

Methods:
#/+/- method_Name(parameter data type/s): returns data type
Description

Class SeedGenerator
Generates base 64 seeds for mazes, and can use seeds to produce maze images

Properties:

height: int <<get/set>>
The number of rows of the maze represented by the seed
width: int <<get/set>>
The number of columns of the maze represented by the seed
maze: Maze
The maze object created by the maze generator
maze_generator: MazeGenerator
The maze generator object used to produce the random mazes and their images
seed: str <<get/set>>
The base 64 seed for the maze
to64: dict
A binary to base 64 conversion table, used for creating seeds from generated mazes
#toBinary: dict
A base 64 to binary conversion table, used when drawing maze images based on base 64 seeds

Methods:

twoDigitNumber(int): int
Used to add a 0 to the front of a single digit number
+ createBase64Seed(): str
Uses a maze generator to create and random maze and draw its image, then produces the base 64 seed for the maze
+ drawMazeFromSeed()
Takes in a base 64 seed and uses the maze generator to draw the correct image for it

Class MazeGenerator

Generates random mazes using recursive backtracking, and creates images to represent them

Properties:

#max_x: int
The number of columns the maze has
#max_y: int
The number of rows the maze has
maze: Maze
The maze object used to store the sequence of nodes generated
nodes: list
A list containing all the possible locations for nodes – ensures that every location will be used
adjacent_coords: tuple
((-1, 0), (1, 0), (0, 1), (0, -1)), allows easy calculation of all nodes adjacent to any node
tile_names: dict
A dictionary relating each 4-digit binary number (every combination of walls and corridors a node can have) to the correct image name

Methods:

+ drawMaze(str): pillow image object
Creates the image for a maze based on a string of binary values – every 4 digits refers to one node
+ recursiveBacktracking(): Maze
Uses recursive backtracking to randomly navigate the sequence of nodes and store them in a maze object

Class Maze

Used to store the list of nodes and a string containing all of the nodes' wall values

Properties:

#max_x: int
The number of columns the maze has
#max_y: int
The number of rows the maze has
node_ list: list
The 2D list of nodes in the maze
binary_list: list
A list of the nodes' binary wall values

Methods:

+ insert(Node)
    Inserts a node into its correct position in the maze list, and adds its binary wall values to the binary list
+ node(list): Node
    Returns a node from the maze at a specified coordinate
+ binary_string(): string
    Joins the binary list into one string and returns it

Class Node

Stores the necessary data for a single node in the maze

Properties:

- pos: list
    The coordinates in the maze that the node exists at
row: int <<get>>
    The row of the maze the node is in
column: int <<get>>
    The column of the maze the node is in
top: int <<get/set>>
    Whether or not there is a wall along the top edge of the node – 1 for a wall, 0 for no wall
bottom <<get/set>>
    Whether or not there is a wall along the bottom edge of the node – 1 for a wall, 0 for no wall
left <<get/set>>
    Whether or not there is a wall along the left edge of the node – 1 for a wall, 0 for no wall
right <<get/set>>
    Whether or not there is a wall along the right edge of the node – 1 for a wall, 0 for no wall

Methods:

wallValueChecker(int): bool
    Ensures that the value being entered for a wall is either a 1 or 0
+ key(): str
    Returns all the wall values as one string – top + bottom + left + right

MazeGenerator.recursive_backtracking() pseudocode

```
A subroutine to generate a random, perfect maze using recursive backtracking

class properties used:
	maze: Maze object
	nodes: empty array

SUBROUTINE recursive_backtracking
	stack: empty array
	next_position <- [1, 1]

	WHILE true DO
		current_position <- next_position
		push current_position to stack

		TRY
			remove current_position from nodes
		IGNORE value error
		ENDTRY

		possible_coordinates: empty array

		FOR coordinate of coordinates adjacent to current_position DO
			IF coordinate is in nodes THEN
				append coordinate to possible_coordinates
			ENDIF
		ENDFOR
```

```
            IF possible_coordinates contains items THEN
                next_position <- random choice from possible_coordinates
                pair <- [next_position, current_position]

                FOR coordinate and index of coordinate in pair DO
                    IF a Node object at coordinate is not in maze THEN
                        node <- new Node object at coordinate
                    ELSE
                        node <- node object from maze at coordinate

                    adjacent_node <- pair[index-1]
                    y_difference <- coordinate[0] - adjacent_node[0]
                    x_difference <- coordinate[1] - adjacent_node[1]
                    IF y_difference > 0 THEN
                        node.top <- '0'
                    ELSE IF y_difference < 0 THEN
                        node.bottom <- '0'
                    IF x_difference > 0 THEN
                        node.left <- '0'
                    ELSE IF x_difference < 0 THEN
                        node.right <- '0'
                    ENDIF

                    insert node into maze
                ENDFOR

            ELSE
                FOR index of each item in stack (backwards) DO
                    check_positon <- stack[index]
                    FOR coordinate of coordinates adjacent to current_position DO
                        IF coordinate is in nodes THEN
                            next_position <- check_positon
                            break for loop
                        ENDFOR
                    ELSE
                        TRY
                            pop from stack
                        IGNORE value error
                        ENDTRY
                        continue to next iteration
                    ENDIF
                    break for loop
                ENDFOR
            ENDIF

            IF stack LENGTH = 0 THEN
                break while loop
            ENDIF

        ENDWHILE
        return maze
ENDSUBROUTINE
```

Converting between base 64 and binary

Conversion between base 64 and binary is used in several places throughout the program. Binary to base 64 is used to turn the binary maze data into a base 64 seed for the URL – but base 64 to binary must also be used to convert a seed into binary data to be stored as nodes in a maze object, both in the python backend and by the JavaScript when the mazes are loaded by the client.

The conversion works by matching every 6 bits to their corresponding base 64 digit, provided by a conversion table – or in the case of a program, a dictionary is used. Because using a dictionary for faster

look up times means that the lookup will be one-way, separate dictionaries are used for converting either way.

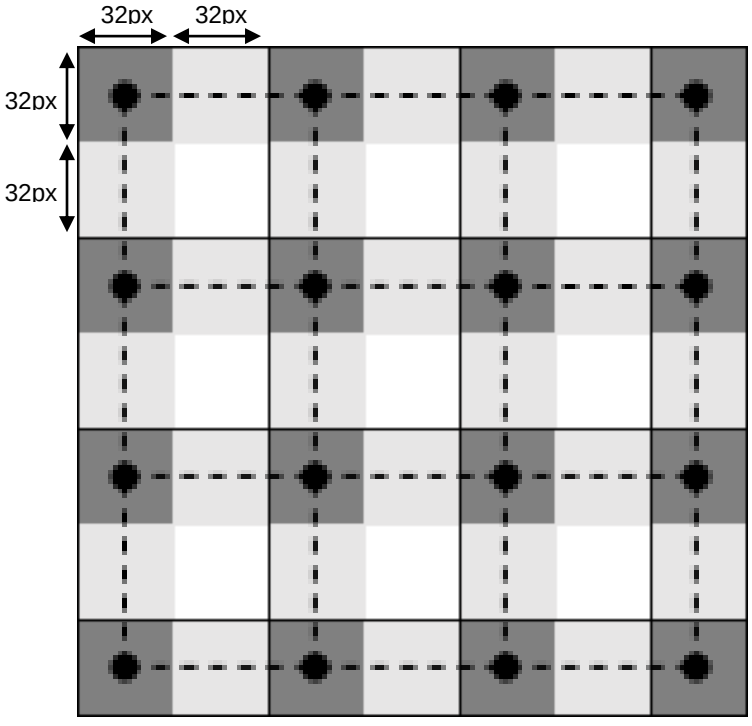
As the Base 64 conversion takes groups of 6 bits, the total length of the binary string must be a multiple of 6. However, we know that the binary string is already a multiple of 4, so if we add 0s until we get a total length which is a multiple of 24, then we must have added a number of 0s that is also a multiple of 4. Because of this, we can divide the number of 0s added by 4, then append that many = to the end of the Base 64 number to indicate how much padding was used – this is necessary to ensure the same number is decoded back into binary, so the program knows how much padding was added to know how much to remove.

Generating the maze images

Either as the maze is generated, or after a given seed is translated, the maze object will have the "binary_string" property. This is then used by the draw_maze function to create the maze image.

Every 4 characters in the binary string represents one node in the maze, and each character represents whether or not there is a wall at the top, bottom, left, or right of the node, in that order, by containing either a 1 (a wall is present) or a 0 (there is no wall). For example, "1001" would have a wall on the top and right, but none on the bottom or left, and "1100" would be a horizontal corridor, having walls on the top and bottom but none on the left or right.

The numbers 0 to 15 in binary all correspond to a particular permutation of walls that a node could have, and each possible node has an image with the correct walls and corridors drawn on in the "/app/static/img/maze" folder. All the images are 32x32 pixels (apart from nodes with walls at the bottom, which are 7 pixels taller – this only affects the full height of the image). So, the maze's nodes can be visualised as such:



The dark grey areas represent positions of nodes in the maze; the light grey areas represent possible positions that edges could occupy; and the white areas represent places where neither nodes nor edges can be.

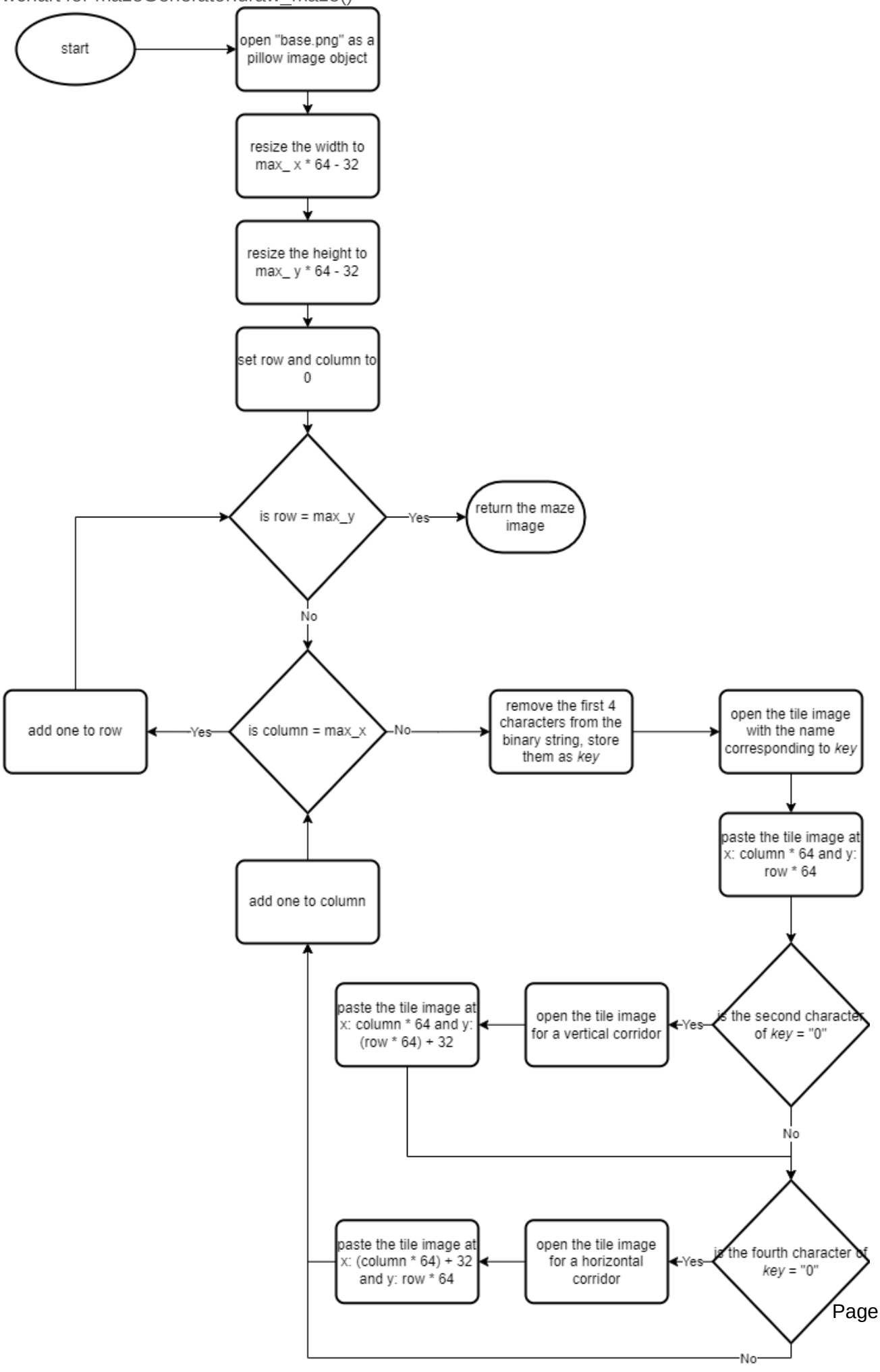
From this diagram it can be calculated that:
- Image height = (rows*32*2) - 32 + 7 (+7 to account for the bottom walls)
- Image width = (columns*32*2) - 32

As Pillow pastes images from the top-left corner, the position that any node should be pasted at is:
- x = column*32*2
- y = row*32*2

Additionally, as is shown by the areas outlined in black, to paste the correct images for the edges, only the light grey areas to the right and bottom of nodes need to be checked to cover every possible position. This can be achieved simply by checking the value of the right and bottom walls for each node: if there is no

wall, paste an image in the correct place. To find this correct position, the only change made to the position calculation is adding 32 to the x or y calculations for horizontal or vertical edges respectively.

Flowchart for mazeGenerator.draw_maze()



Flask Webapp

Webapp endpoints:

Path	HTTP methods	URL format	Returns	Authentication	Errors	Purpose
/	GET	No data	index.html	public		Serves as the homepage for the web app
/play	GET	?height=*int*&width=*int*&maxEnemies=*int*&maxLocks=*int*	Redirects to /play/[seed]	public		Takes values to generate a new, random maze with.
/play/[seed]	GET	[seed]: base 64 seed for a maze	play.html	public	500 - if the seed is invalid	Serves as the playable level for any specific maze
/gamecomplete	GET	No data	gamecomplete.html	public	401 - if the user has not completed any mazes	Serves and the end of the game
mazeimages/[image name]	GET	[image name]: name of image to be retrieved	Maze image	public	404 - if the image does not exist	Serves as a way of accessing the randomly generated maze images

How does views.py handle http requests

Flask uses decorators to determine which function to run for a given request. There are some functions of views.py that do not link directly to web page routes:

“save_maze_image(maze_image)” takes a pillow image object and saves it in a file which name includes the current datetime value. This name is then saved to the session dictionary with the key ‘image name’

“add_maze_to_session(seed)” takes a seed and appends it to the list of mazes attempted in the current session

“check_side_length(length)” makes sure a given length is within the acceptable bounds for a maze, at least 3 and at most 25

The following examples of pseudocode make use of these functions along with the flask “session” dictionary, which is a dictionary provided by the flask library to store data for a single session only.

Function for route `/`

```
SUBROUTINE index()
	return flask render_template("public/index.html")
ENDSUBROUTINE
```

Function for route `mazeimages/[image name]`

Serves the image for a maze, will result in a 404 status if the file name given does not exist

```
SUBROUTINE get_maze_image(file_name)
	return flask send_from_directory(root path for "/app/mazeimages", file_name)
ENDSUBROUTINE
```

Function for route `/play`

The URL takes a query containing up to 4 parameters, height, width, maxEnemies and MaxLocks. The function checks the parameters entered – which ones exist and if their values are valid – before generating a seed based on the width and height, and redirecting to the page for that seed.

```
SUBROUTINE play_with_size()
	args <- dictionary of flask request arguments

	TRY
		IF key 'maxEnemies' is in args THEN
			session['max enemies'] <- integer(args['maxEnemies'])
		ELSE
			session['max enemies'] <- -1
		ENDIF

		IF key 'maxLocks' is in args THEN
			session['max locks'] <- integer(args['maxLocks'])
		ELSE
			session['max locks'] <- -1
		ENDIF
		IF key 'height' is in args THEN
			seed_generator.height <- checkSideLength(integer(args['height']))
		ELSE
			seed_generator.height <- 3
		ENDIF

		IF key 'width' is in args THEN
			seed_generator.width <- checkSideLength(integer(args['width']))
		ELSE
			seed_generator.width <- 3
		ENDIF
	CATCH err <- value error
		OUTPUT "bad request", err type, err message
		return flask redirect("/", status <- 400)
	ENDTRY

	maze_image <- seed_generator.create_base_64_seed()
	save_maze_image(maze_image)
	add_maze_to_session(seed_generator.seed)

	return flask redirect("/play/" + seed_generator.seed)
ENDSUBROUTINE
```

Function for route `/play/[seed]`

Tries to draw a maze image if the seed given is not already contained by the seed generator, i.e. if the user has been redirected from `/play`

```
SUBROUTINE play_from_seed(seed)
	IF seed_generator.seed != seed THEN
		TRY:
			seed_generator.seed <- seed
			maze_image <- seed_generator.draw_maze_from_seed()
		CATCH err <- (index error, key error, value error, attribute error)
			OUTPUT "invalid seed", err type, err message
			return flask redirect("/", status <- 500)

	ENDIF

	save_maze_image(maze_image)
	add_maze_to_session(seed)

	IF key 'game complete' is not in session THEN
		session['game complete'] <- 0
	ENDIF

	IF session['game complete'] = true THEN
		level <- custom
	ELSE
		level <- string(length of session['maze list'])
	ENDIF

	IF 'max enemies' is in session THEN
		enemies <- session['max enemies']
		remove 'max enemies' from session
	ELSE
		enemies <- -1
	ENDIF

	IF 'max locks' is in session THEN
		locks <- session['max locks']
		remove 'max locks' from session
	ELSE
		locks <- -1
	ENDIF

	return flask render_template("/public/play.html", level <- level, maxEnemies <- enemies,
maxLocks <- locks, mazeImage <- session['image name'], mazeSeed <- seed_generator.seed. gameComplete <-
session['game complete'])
ENDSUBROUTINE
```

Function for route `/gamecomplete`

```
SUBROUTINE game_complete()
	IF 'maze list' is not in session THEN
		flask abort(status <- 401)
	ENDIF

	session['game complete'] <- 1

	return flask render_template('public/gamecomplete.html', mazeList <- session['maze list']
ENDSUBROUTINE
```

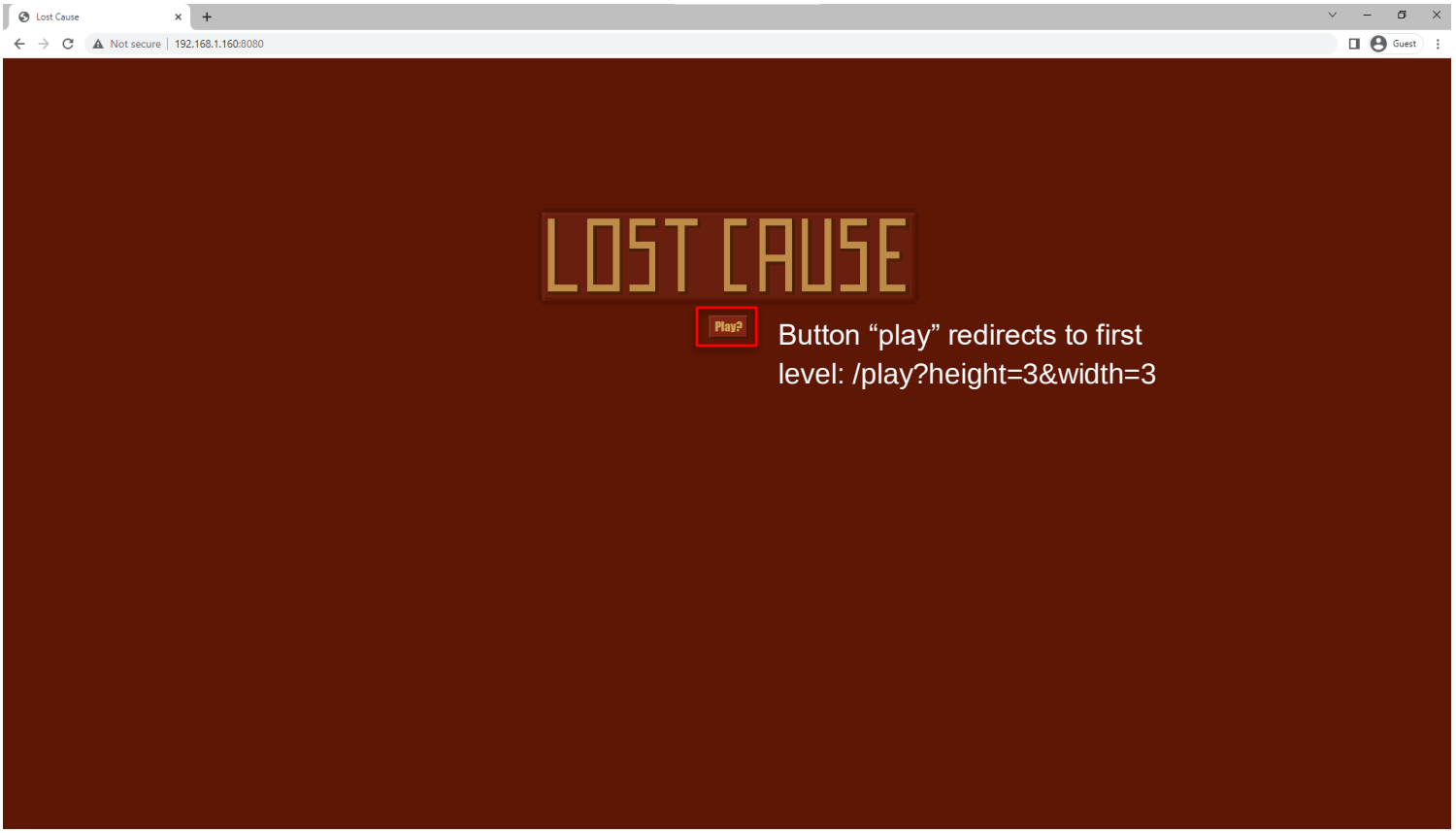
The Frontend

Webpage states:

The Following web pages makes use of the jinja templates public.html and level_template.html to form the basis of the html template.

`/` :

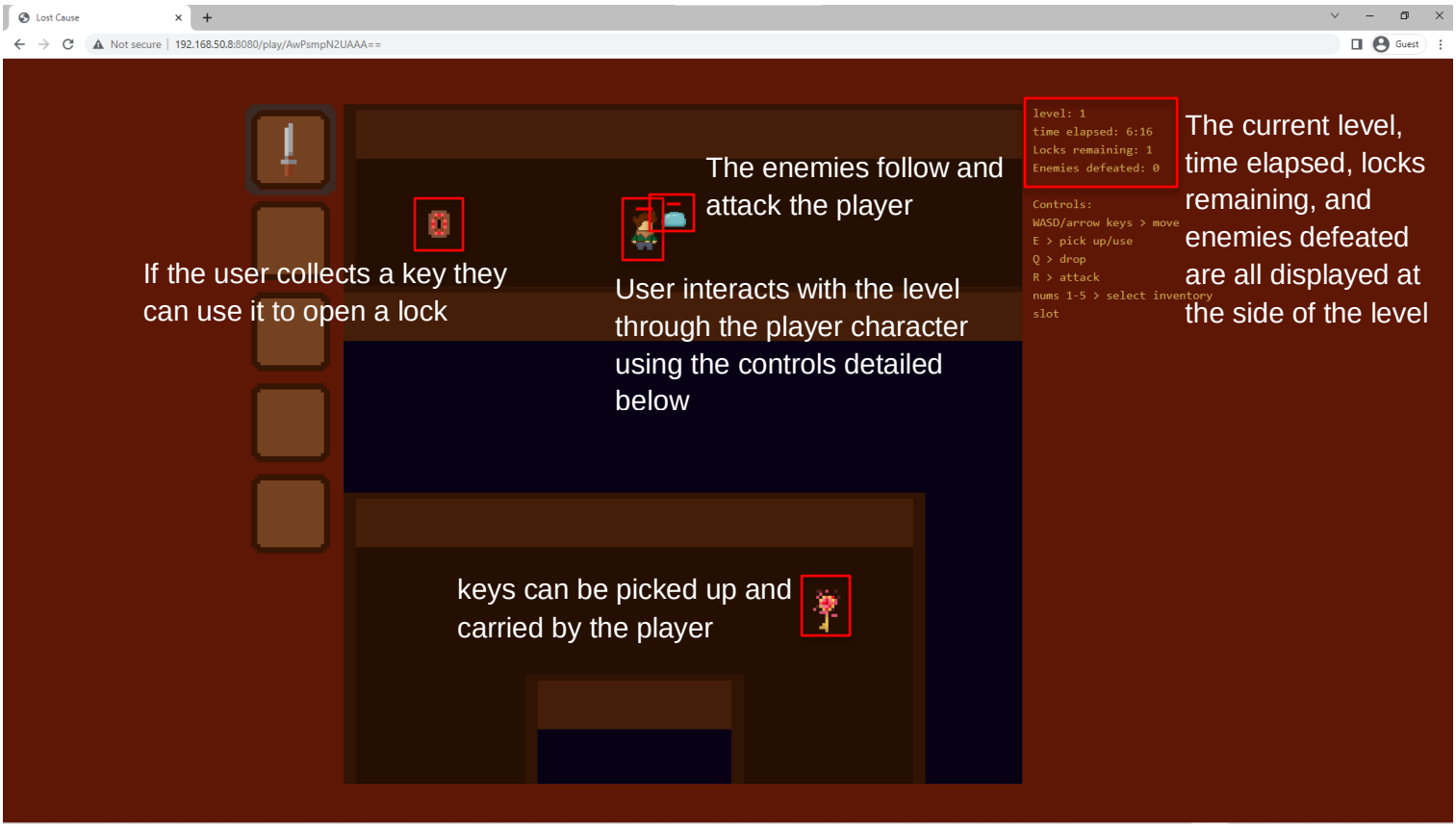
Uses index.html



`/play/[seed]`:

Uses level.html

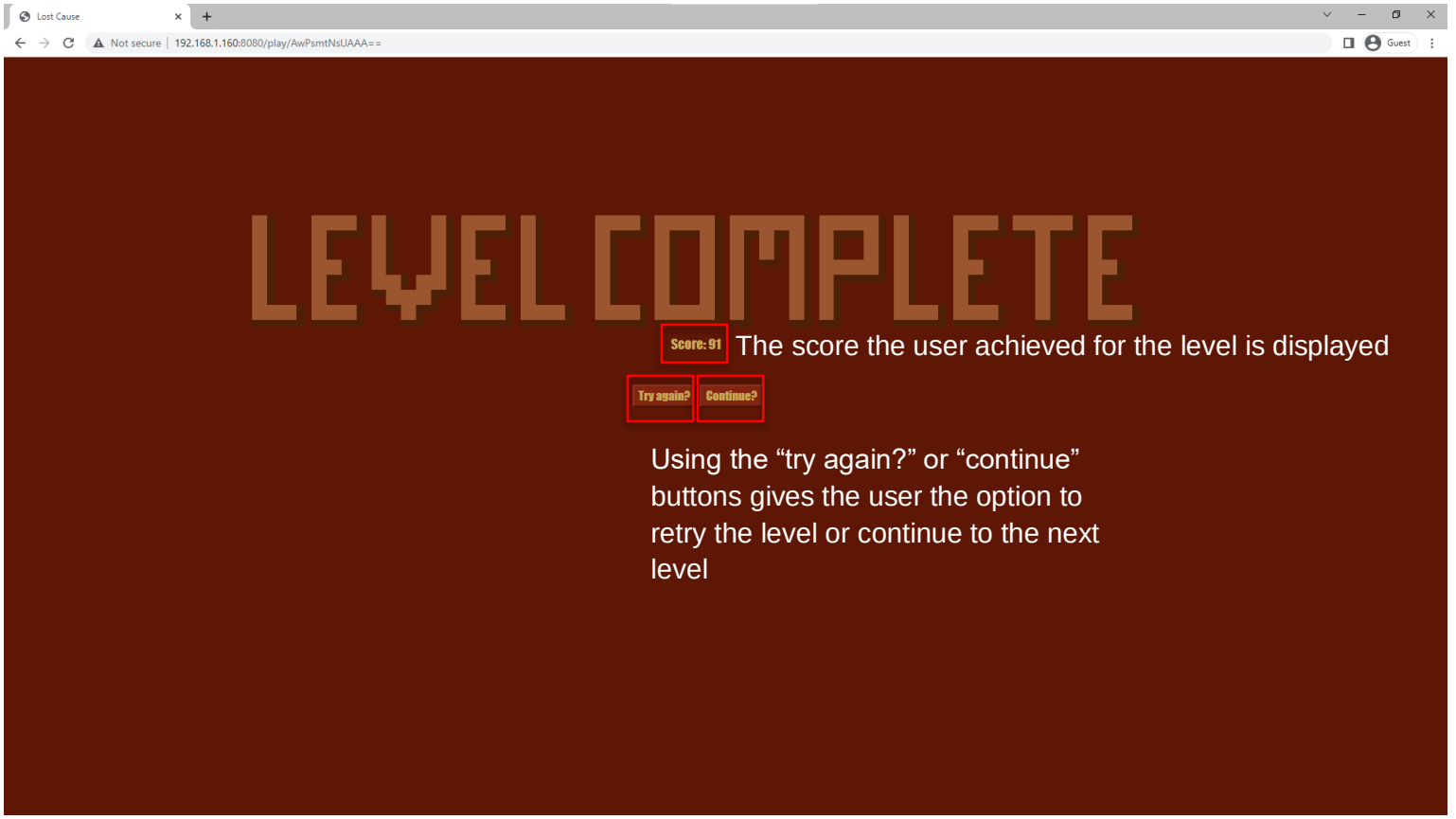
.level#shown .end-screen#hidden - when the level is being played



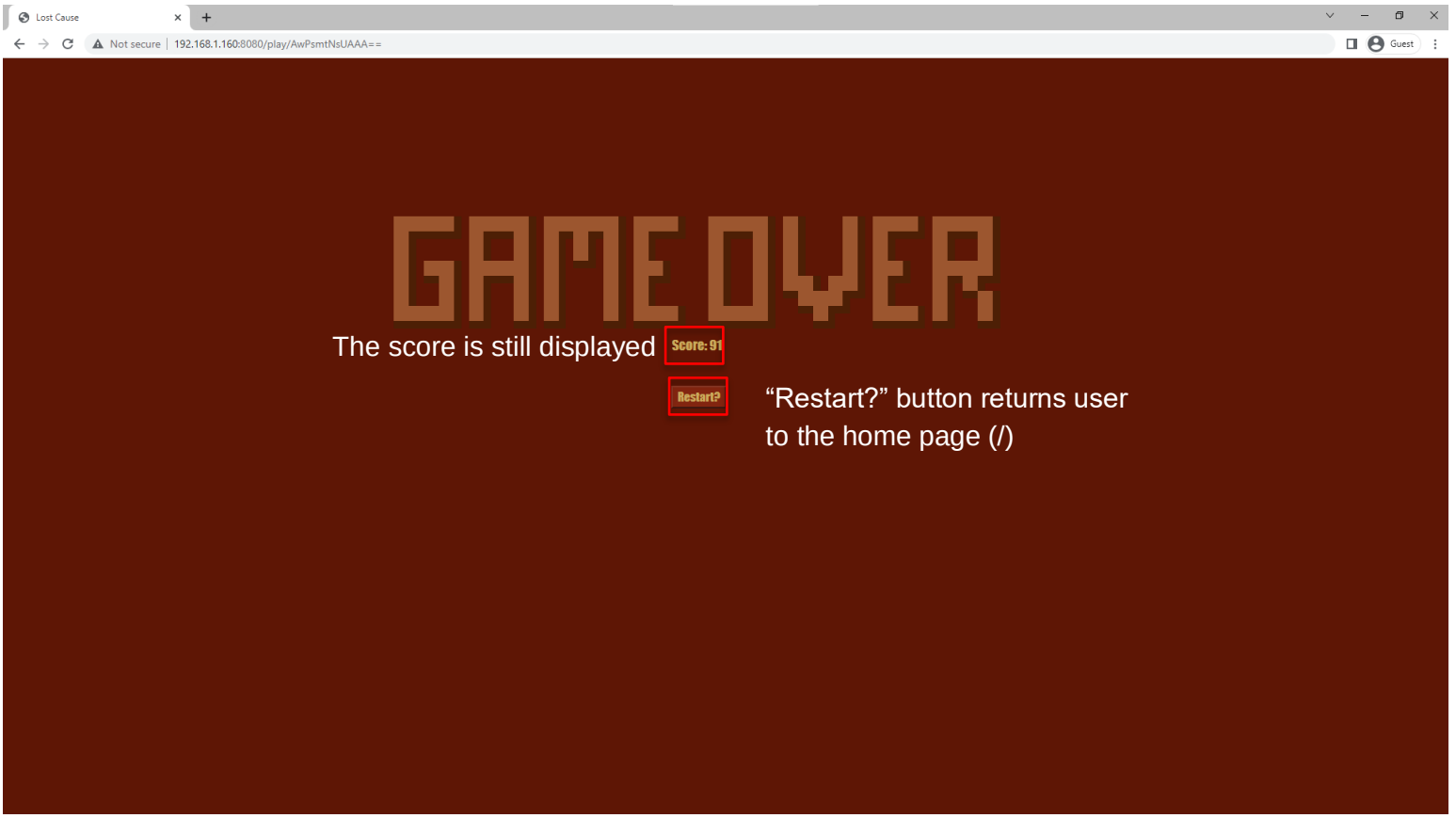
Level controls:

- W or up arrow: moves up
- A or left arrow: moves left
- S or down arrow: moves down
- D or right arrow: moves right
- E: pick up item or open lock
- Q: drop item
- R: attack enemy
- Numbers 1-5: selects corresponding inventory slot
-

.level#hidden .end-screen#shown - if the level has been completed successfully:

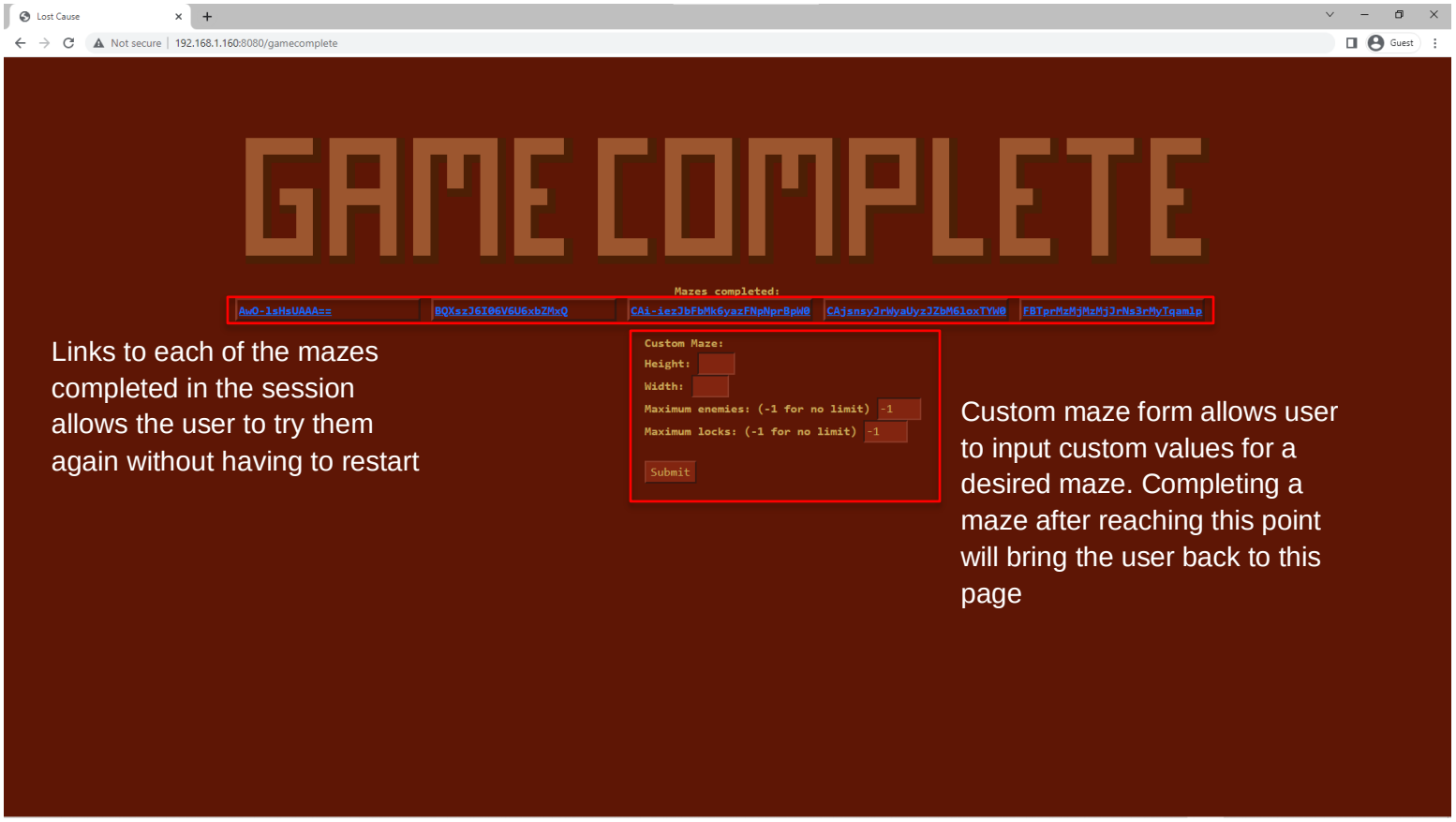


.level#hidden .end-screen#shown - if the level has been failed:



`/gamecomplete`:

Uses gamecomplete.html

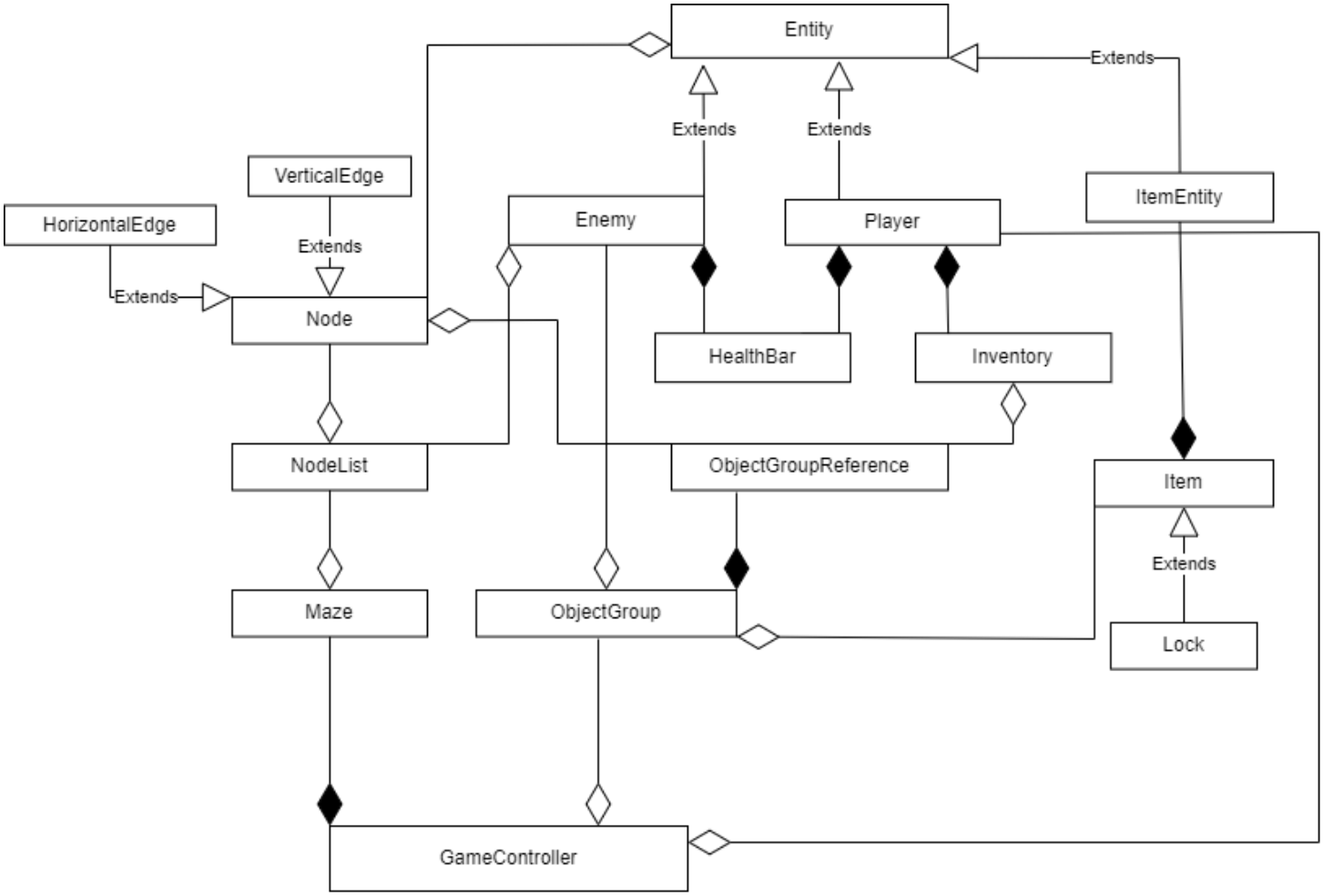


Links to each of the mazes completed in the session allows the user to try them again without having to restart

Custom maze form allows user to input custom values for a desired maze. Completing a maze after reaching this point will bring the user back to this page

app.js

UML diagram for app.js



app.js class descriptions

Format:

: protected, + : public, - :private

Class *ClassName* extends *Parent*

Description

Fields:

#/+/- fieldName: data type

Description

Methods:

#/+/- methodName(parameter data type/s): returns data type

Description

Class *GameController*

Handles the set-up for each level, controls the game loop

Fields:

gameOver: bool
Tells the controller whether not the game has ended
+ itemGroup: ObjectGroup
Contains all the items in the game
+ lockgroup: ObjectGroup
Contains all the locks in the game
+ enemyGroup: ObjectGroup
Contains all the enemies in the game
+ heldDirections: arr
List of all the directions currently being held by the user
+ activeInventorySlot: int
The currently selected inventory slot
+ map: DOM element
A reference to the map element of the DOM
+ endScreen: DOM element
A reference to the end-screen element of the DOM
+ level: DOM element
A reference to the level element of the DOM
+ timeElapsed: int
How much time has passed since the game loaded
timerStatus: DOM element
A reference to the element for the timer in the DOM, shows user how much time has passed
locksStatus: DOM element
Shows user how many locks are remaining
enemiesStatus: DOM element
Shows user how many enemies they have defeated
+ player: Player
The Player object
+ maze: Maze
The Maze object
+ deadEndPositions: arr
Contains all the dead ends of the maze, used to determine where locks and keys are spawned
+ enemySpawnPositions: arr
Contains a list of the positions that enemies should be spawned
+ root: DOM element
A reference to the root DOM element, used to set CSS variables for the height and width of the maze image
+ imgWidth: int
The width of the maze image
+ imgHeight: int
The height of the maze image

Methods:

updateGameStatus(): void
Called every frame to update the timer, locks opened and enemies defeated.
+ setActiveInventorySlot(int): void
Called when the user changes the active inventory slot
+ checkWinCondition(): bool
Checks if all the locks have been opened, i.e. if the game has been won
+ spawnEnemies(): void
Spawns enemies at the positions specified by enemySpawnPositions
+ spawnPlayer(): void
Spawns the player at [1,1]
+ defineMaze(): void
Creates a new maze using the seed and uses the binary string to populate the maze's nodelist with nodes, also records dead ends and determines enemy spawning positions
+ determineLockAndKeyPositions(): void
Uses the dead ends to determine where locks and keys should be spawned, and spawns them
+ defineMazeProperties(): void
Sets the –map-width and –map-height CSS variables and imgWidth and imgHeight fields to the width and height of the maze image
+ gameEnd(bool): void
Called when the game ends, takes true if the game was won and false if the game was lost - displays the game-over element with the appropriate message
+ gameLoop(): void
Controls what happens every frame
+ step(): void
Requests animation frames, calls gameLoop()

Class *Maze*

Stores and handles all the nodes in the maze

Fields:

+ seed: str
  The base 64 seed for the maze
usedEdges :arr
  Edges currently used to contain an item
+ nodes: NodeList
  The NodeList containing all the nodes in the maze
+ height: int
  The number of rows of nodes
+ width: int
  The number of columns of nodes
+ binaryString: str
  The binary representation of the seed

Methods:

+ checkTileInMaze(int, int): bool
  Checks if a tile at a given coordinate exists in the maze (used to check nodes and edges)
+checkUsedEdge(Node): void
  Check if an edge is currently being used to contain an item
+ getTile(int, int): Node
  Gets the node or edge object for a specified location
+ output(): void
Prints the adjacencyList to the console

Class *Node*

Used to store the properties of each node in the maze

Fields:

+ type: str
  Whether the node is a node or an edge
+ x: int
  The column of the maze the node is in
+ y: int
  The row of the maze the node is in
+ top: int
  Whether the node has a top wall or not
+ bottom: int
  Whether the node has a bottom wall or not
+ left: int
  Whether the node has a left wall or not
+ right: int
  Whether the node has a right wall or not
+ contains: ObjectGroupReference
  A reference to the item on the tile, undefined if there is none

Methods:

+ position(): obj
  Returns an object containing the x and y positions of the node

Class *HorizontalEdge* extends *Node*

Defines properties of horizontal edges, which are always identical

Fields:

+ type: str
  The type is always ‘edge’

Methods:

None

Class *VerticalEdge* extends *Node*

Defines properties of vertical edges, which are always identical

Fields:

+ type: str
  The type is always ‘edge’

Methods:

None

Class *Item*

Stores the properties and entities for items

Fields:

+type: str
  The type of the item
+ entity: ItemEntity
  The entity for the item
+ id: int
  The unique identifier for the item

Methods:

+ place(int, int): void
  Places the item entity at a specific position on the map
+ take(): void
  Removes the item entity from the map
+ update(): void
  Calls the item entity’s update function

Class *Lock* extends *Item*

A specific kind of item, which cannot be picked up

Fields:

+ locked: bool
  Whether or not the lock is locked

Methods:

+ unlock(): void
  Sets the value of locked to false, and removes the lock from the map

Class *ObjectGroup*

Stores a list of all the objects of a specific type

Fields:

+ objectType: str
  The type of the objects being stored
+ objectList: arr
  The list of objects

Methods:

+ get(int): obj
  Returns the object at a specified index
+ push(object): ObjectGroupReference
  Adds a new object to the list, returning a reference to the object

Class *ObjectGroupReference*

A reference to a specific item in an object group

Fields:

+ objectType: str
    The type of the object
index: int
    The index of the object in the object group's list
objectGroup: ObjectGroup
    The object group the object is in

Methods:

+ getSelf(): obj
Returns the object that is being referred to

Class *Entity*

An object that is on the game's map

Fields:

+ x: int
    The x coordinate of the entity relative to the map
+ y: int
    The y coordinate of the entity relative to the map
speed: int
    Default value for how far the entity can travel every frame
prevTile: obj
    The tile the entity was in on the previous frame
currentTile: obj
    The tile the entity is in on the current frame
tileOrigin: obj
    The x and y coordinates of the top-left corner of the tile the entity is in
+ moveDirections: arr
    The directions the entity is currently trying to move in
attackCooldown: int
    Default value for the time the entity has to wait between attacks
attackDamage: int
    Default value for the damage the entity deals when attacking

Methods:

+ roundTileCoord(int): int
    Rounds a coordinate down to the nearest half
+ determineCurrentTile(): void
    Determines the tile of the maze the entity is currently in
+ determineTileOrigin(): void
    Determines the tile origin of the current tile
+ checkCollision(int, int): void
    Checks if the entity will collide with a wall if it moves to a specified position
+ move(): void
    Moves the entity in the direction specified by moveDirections
+ getCurrentTile(): Node
    Returns the node object for the current tile the entity is in
+ attack(Entity): void
Attacks a specified entity

Class *ItemEntity* extends *Entity*

The entity for an item

Fields:

+ self: DOM element
    The items element to be added to the document

Methods:

+ determineCurrentTile(): void
    Determines the tile of the maze the entity is currently in, has slightly different functionality to the parent method
+ determineTileOrigin(): void
    Determines the tile origin of the current tile
+ place(): void

Adds the DOM element to the document and puts the entity in the correct position
+ updatePosition(): void
Updates the position of the entity, in case the pixel size has changed
+ spawn(int, int): void
Places the entity based on tile coordinates
+ move(int, int): void
Places the entity based on x and y values
+ remove(): void
Removes the entity from the map

Class *Enemy* extends *Entity*

The hostile, non player character

Fields:

+ id: int
The unique identifier for the enemy
speed: int
How far the enemy can move every frame
range: int
The radius of the area the enemy can pathfind to the player in
maxHealth: int
The amount of health the enemy starts with
health: int
How much health the enemy has
attackCooldown: int
How long the enemy has to wait between attacks, prevents player from being reduced to 0 health instantly, and allows time for the attack animation to play
attackDamage: int
How much damage the entity does per attack
path: arr
A list of directions the entity should follow to reach the player
target: obj
The exact x and y values of the enemies target (the player) - both set to -1 if the enemy is not in the same tile as the player
targetTile: obj
The tile that the enemies target is in - what that path found is to
healthBar: HealthBar
Object for the enemies's health bar
+ self: DOM element
The element for the enemy on the map

Methods:

+ spawn(int, int): void
Spawns the enemy in a specified tile
+ damage(int): void
Called when the enemy is attacked, damages the enemy
+ pathFind(): void
Finds a path to follow to reach the player
+ move(): void
Moves the enemy either along the path or directly towards the player
+ attack(): void
Attacks the player

Class *Player* extends *Entity*

The character that the user controls and interacts with the game through

Fields:

speed: int
How far the player can move every frame
maxHealth: int
The amount of health the player starts with
health: int
The amount of health the player currently has
attackCooldown: int
How long the player has to wait between attacks - allows the attack animation to play

attackDamage: int
How much damage the player deals in one attack
+ inventory: Inventory
The object to store all the items the player is carrying
+ self: DOM element
The element for the player on the map
healthBar: HealthBar
Object for the player's health bar
+ enemiesKilled: int
How many enemies the player has defeated - displayed in the game status
+ locksOpened: int
How many locks the player has opened - displayed in the game status

Methods:

+ executeCommand(str): void
Called when a command key is pressed, runs the function corresponding to the command
+ interact(): void
Will pick up an item if there is an item to pick up, or unlock a lock if the player is holding a key and a lock is present
+ drop(): void
Drops the item in the currently active inventory slot if there is space in the tile
+ damage(int): void
Called when the player is attacked, damages the player
+ move(): void
Moves the map in the opposite direction to the player unless the player is at the edge of the map - gives the impression of camera following
+ attack(): void
Attacks the nearest enemy if the player is holding a weapon

Class *HealthBar*

Controls the health bar element for a specific entity

Fields:

maxHealth: int
The starting health of the entity, used to calculate the relative size the health bar should be
self: DOM element
The element for the health bar, a child of its entity

Methods:

+ update(int): void
Updates the size of the health bar, called when the entity takes damage

Class *Inventory*

Contains all the items the player is currently holding

Fields:

size: int
The maximum number of items the inventory can hold
slotsAvailable: int
How many free slots there are
contents: arr
A list of object group references to items that are in the inventory

Methods:

#setDocumentInventorySlot(int, str): void
Changes the id of a specified slot in the document to the type specified
+ getItemFromSlot(int): ObjectGroupReference
Returns the object group reference for the item in a specific slot
+ insertItem(ObjectGroupReference, int): void
Adds an item to the next available slot in the inventory
+ removeItem(int): ObjectGroupReference
Removes an item from a specified slot, and returns the reference
+ checkSlotsAvailable(): bool
Checks if there are any slots available, i.e. if the player can pick up more items or not

Calculating lock spawning

As the mazes in the game do not have a physical exit, it seemed logical to place the objectives (locks and keys) in dead ends, so a player would have to explore the whole, or most of, any maze to complete it.

To determine whether or not dead ends would be suitable, I used the recursive backtracking algorithm created for the prototype of the solution to create 100,000 mazes for each square maze from side length 3 to 25. For every maze generated, the number of dead ends is counted, and recorded in a frequency table stored in a dictionary. These values were then used to calculate the mean and standard deviation of the dead ends for each size of maze, and the mean percentage of nodes in each maze that are dead ends.

The following table shows the data gathered

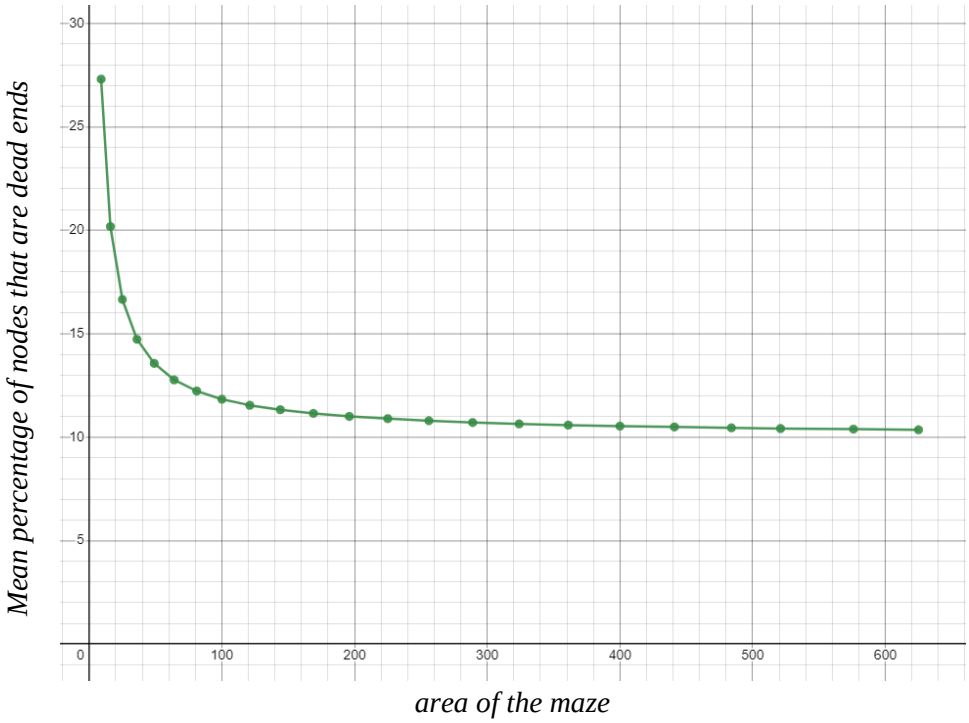
(Note, the theoretical values for a 1x1 and 2x2 recursive backtracking maze are also shown)

Height x width	Number of nodes	Mean number of dead ends	Standard deviation of dead ends	Mean percentage of nodes that are dead ends
1x1	*1*	*1*	*0*	*100*
2x2	*4*	*2*	*0*	*50*
3x3	9	2.45746	0.4981870616545544	27.305111111111113
4x4	16	3.22892	0.7227002377196224	20.18075
5x5	25	4.16384	0.9452070960376868	16.65536
6x6	36	5.30177	1.119975386827762	14.727138888888891
7x7	49	6.648	1.3071021383197272	13.56734693877551
8x8	64	8.17109	1.4881727762259331	12.767328124999999
9x9	81	9.90469	1.659351079157149	12.228012345679012
10x10	100	11.83439	1.847426135979455	11.83439
11x11	121	13.96563	2.029780456872118	11.541842975206611
12x12	144	16.3108	2.2166829633486143	11.326944444444445
13x13	169	18.84168	2.381708373751915	11.148923076923078
14x14	196	21.56218	2.5770125431592183	11.001112244897959
15x15	225	24.50987	2.7517526384288256	10.893275555555554
16x16	256	27.63495	2.939664691338121	10.79490234375
17x17	289	30.94035	3.123026076980471	10.706003460207612
18x18	324	34.47706	3.3032883247454916	10.64106790123457

19x19	361	38.21026	3.4703617581457067	10.584559556786703
20x20	400	42.14319	3.6547101969787095	10.5357975
21x21	441	46.2752	3.8551037547645746	10.493242630385486
22x22	484	50.59264	4.026961364403647	10.453024793388431
23x23	529	55.10529	4.19894320227134	10.416879017013231
24x24	576	59.83733	4.382347369971899	10.388425347222222
25x25	625	64.74393	4.58027053295978	10.3590288

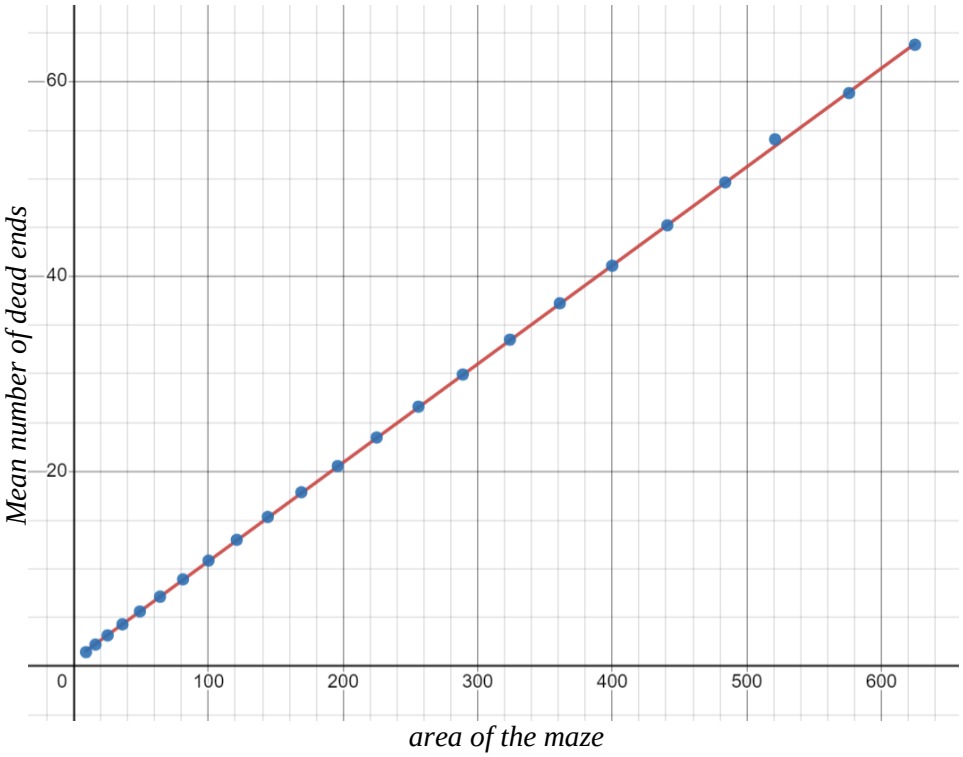
The following graph shows the relationship between the mean percentage of nodes in a maze and the side length of the same maze. (*note: the general shape of the graph is the same if the maze area is used instead of the side length)*

As is shown by both the graph and the data in the table, the percentage of nodes that are dead ends levels out, to approx. 10%.



The following graph shows the relationship between the mean number of dead ends in a maze and the area of the maze. As you can see, this is a linear relationship, following the graph of:

$$y = \frac{x}{9.88} + \frac{2}{3}$$



Overall, this data shows that the number of dead ends will increase proportionally to the size of the maze, whilst also taking up a relatively constant proportion of the maze (for mazes with larger areas). This means that using dead ends to place objectives will satisfy the requirement that the number of objectives in each level should increase as the size of the maze does, whilst not becoming too difficult to realistically complete for larger mazes, which by nature would be harder and take longer to complete anyway.

Knowing that the dead ends are going to be used to place objectives in, we can check each node object created when the seed is being used to create the maze object's node list and record the positions of dead ends to subsequently spawn locks and keys in a list.

When spawning the locks and keys, we need to consider the fact that there should be the same number of keys as there are locks. This means that if the total number of dead ends in the maze is odd, we cannot use one. As the node the player starts in will always be a dead end (as that is where the algorithm starts), this would be the most appropriate node to not use if there is an odd number of dead ends, so the player is not potentially forced to walk back through the entire maze. Taking this into account, and knowing that this node will also the at the start of the list containing all the dead end positions, we can iterate over the list backwards, down to the index at the remainder of dividing the number of dead ends by 2 (will be 0 for even, if the start node should be used, and 1 for odd, if the start node shouldn't be used).

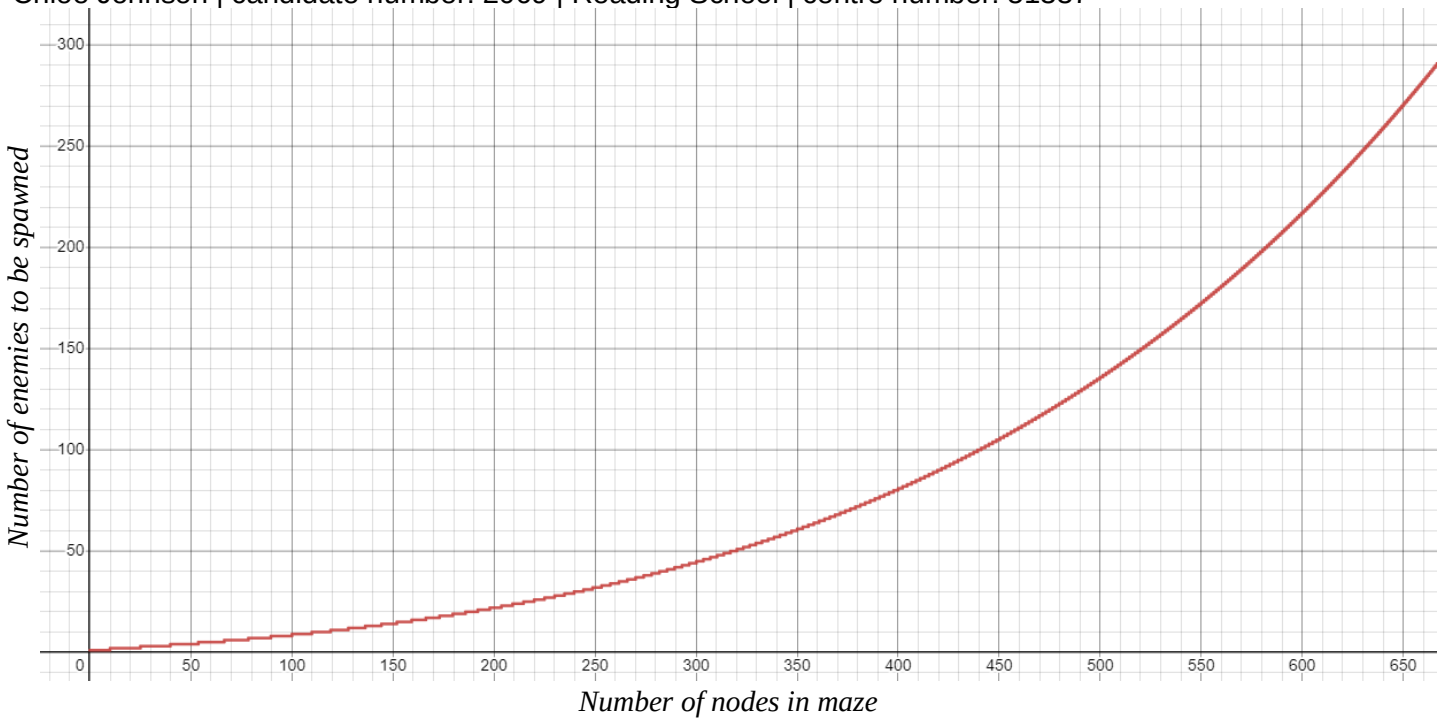
Also, by spawning locks on even indexes of the dead-end positions list, and keys on even indexes, we maintain that the number of locks and keys are the same, whilst also relatively evenly spreading out looks and keys - as opposed to spawning keys for the first half of indexes and locks for the second, which could cause a grouping effect.

Calculating enemy spawning

To determine where the enemies should spawn in the maze, we first need to calculate how many enemies should spawn in each maze.

In order to achieve a reasonable number of enemies per level, whilst maintaining the objective that the number of enemies should increase as the size of the maze does, the following formula was designed. As shown by the graph, as the size of the maze increases so does the number of enemies per node in the maze. However, this value is never above 0.5 for all valid areas (9 to 625), for example for a maze of area 625 (a 25x25 maze), there would be 243 enemies, equating to 0.389 (3.s.f.) enemies per node in the maze

$$\text{floor}\left(3\frac{\sqrt[2]{\text{height} \cdot \text{width}}}{5}\right)$$



Now that we know how many enemies are going to be present in each maze, we need to work out where to spawn them.

The best way to do this in a random maze is to evenly space the enemies throughout the maze. This still gives the illusion that they are randomly placed, due to the inherent randomness of the maze, but avoids the need to actually place the enemies randomly throughout the maze. To find the space (or number of nodes) between each enemy we can simply divide the area (total number of nodes) of the maze by the number of enemies that we want to place.

We don't want an enemy to spawn on top of the player, however, so we can simply offset the spawning by half of the space to leave between each enemy. The sequence created by following this method, and the subsequent formula for working out – or checking – the which nodes should contain enemies is as follows:

n: number of the enemy
x: space between enemies
i: index of node to spawn enemy in

n:	1	2	3	4
i:	$\frac{x}{2}$	$\frac{x}{2}+x$	$\frac{x}{2}+2x$	$\frac{x}{2}+3x$

$$i = \frac{x}{2} + (n-1)x$$

$$i = x\left(\frac{1}{2} + n - 1\right)$$

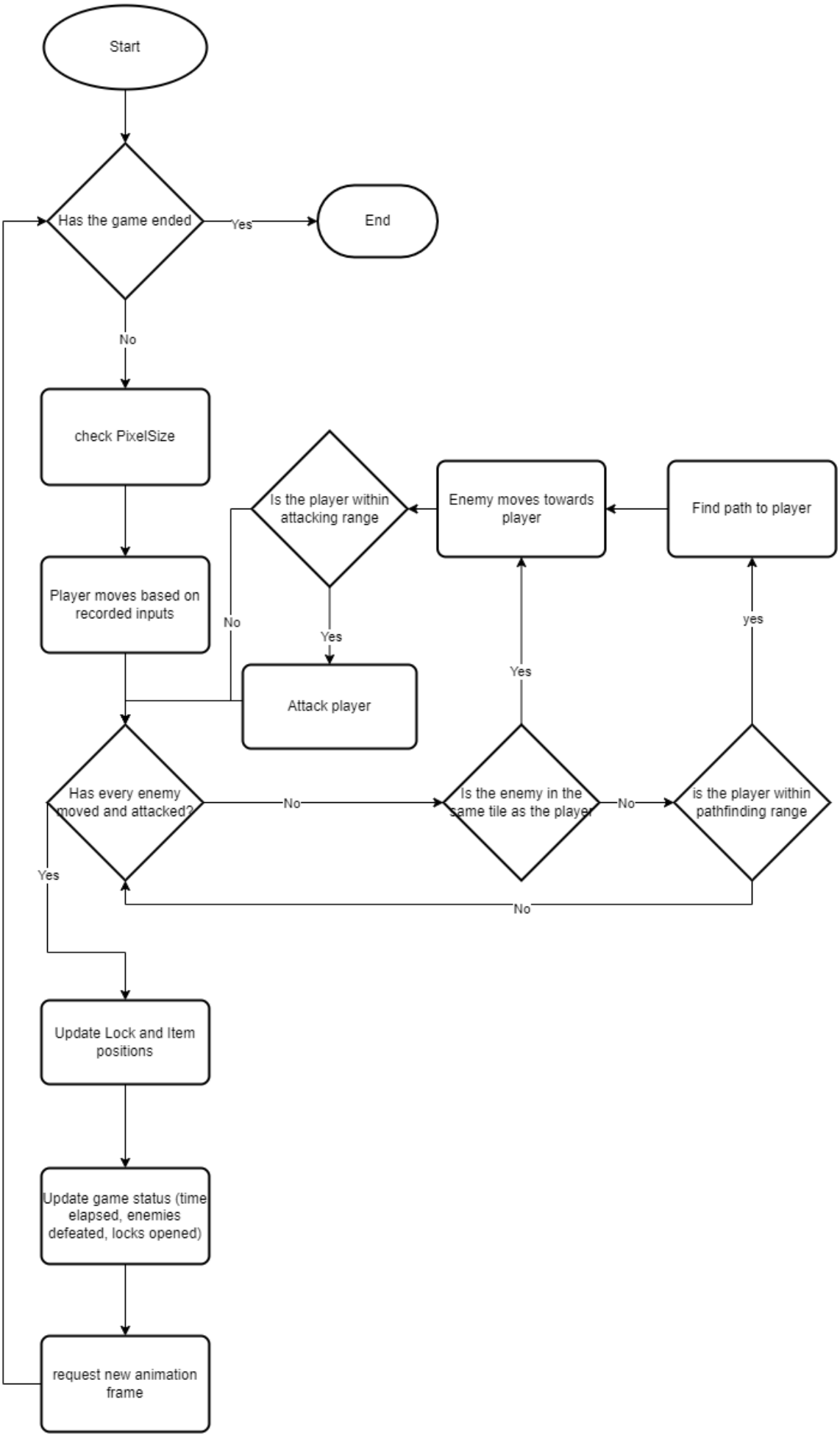
$$i = x\left(n - \frac{1}{2}\right)$$

To make this even simpler to code, we can represent $(n - ½)$ as some value m and start the "enemy count" at 0.5, adding one for each enemy added. Doing this, the formula becomes: $i = mx$

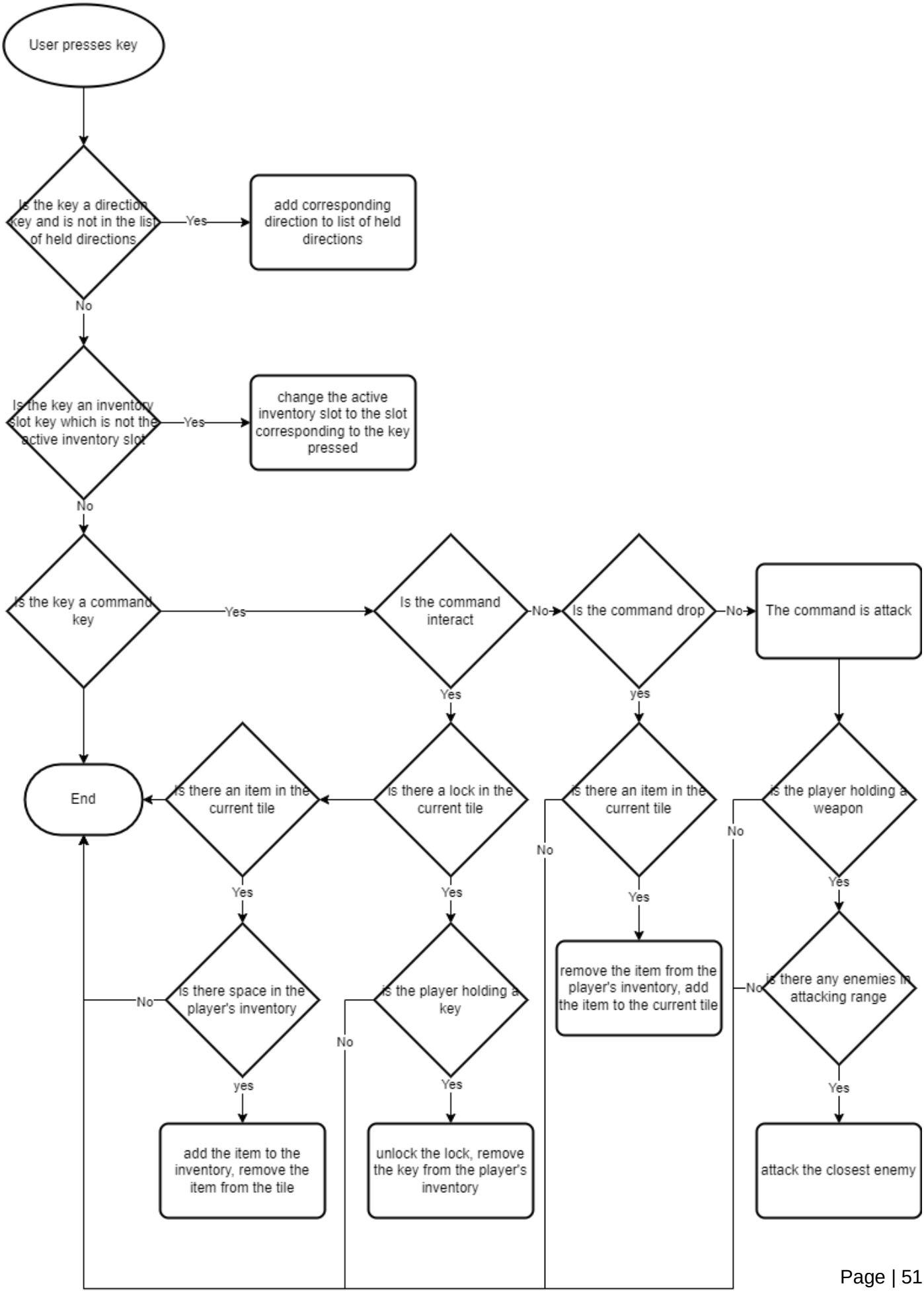
The following pseudocode demonstrates how important mathematical models for equations like this can be, so be able to simplify the equation as much as possible to greatly improve the readability and ease of understanding of the code

```
index <- (enemySpawnSpacing/2) + ((enemyCount - 1) * enemySpawnSpacing)

index <- enemySpawnSpacing * (enemyCount - 1/2)

index <- enemySpawnSpacing * enemyCount
```

Game processes: a flowchart of gameController.gameLoop()


Start
Has the game ended
Yes
End
No
check PixelSize
Player moves based on recorded inputs
Is the player within attacking range
Enemy moves towards player
Find path to player
No
Yes
Attack player
Yes
yes
Has every enemy moved and attacked?
No
Is the enemy in the same tile as the player
No
is the player within pathfinding range
Yes
No
Update Lock and Item positions
Update game status (time elapsed, enemies defeated, locks opened)
request new animation frame

Asynchronous player action: The processes that occur when the user presses a key



PixelSize

As the size of the browser window changes, the best way of scaling pixelated images is by using constant values, as opposed to percentage scaling. Because of this, the CSS variable in level.css –pixel-size is set to a value between 1 and 4 depending on the width of the browser window. This pixel size variable is then used throughout the CSS - all the values determining height, width and margins are multiplied by the pixel size - to control the scale of the different elements.

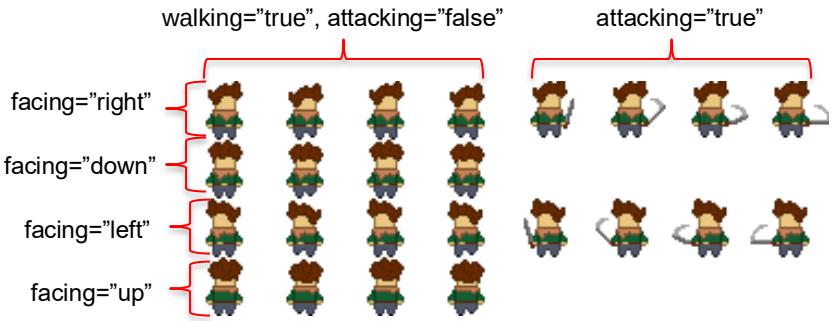
Width (px)	Pixel size
0-750	1
750-1000	2
1000-1400	3
1400+	4

pixelSize is also a global variable of app.js and is used when the program calculates the positions of entities and the map.

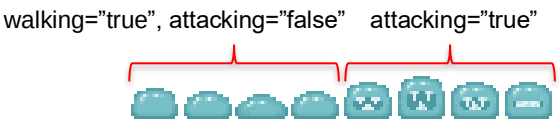
CSS Animations, Entity Movement and Attacking

To give the illusion of movement of the player character and the enemies, sprite sheets are used. Each sprite sheet contains static images for each frame of the movement animation, with different rows for each direction, if necessary.

Player sprite sheet:



Enemy sprite sheet:



The frame shown is determined by three HTML attributes of the character: facing, attacking, and walking - and two attributes of the enemy: walking and attacking.

For the character, the facing attribute determines the row that is animated, but "down" and "up" are only used when walking is "true" and attacking is "false"

For both the character and enemy, the walking animation (when walking is "true" and attacking is "false") continuously loops the sprite sheet over its first half, taking 0.6 seconds for every 4 frames. The attacking animation for both occurs once when attacking is set to "true", animating the latter half of the sprite sheet in 0.2 seconds.

The values of the attributes walking, attacking and facing are controlled by the move() and attack() functions of the entity class, which are called by the player's and enemy's move() and attack() functions when needed.

Entity.move() pseudocode:

```
SUBROUTINE move()
	IF moveDirections contains items THEN
		originalX <- x
		originalY <- y

		call determineCurrentTile

		FOR direction of moveDirections DO
			IF direction = "right" THEN
				x <- x + speed
			IF direction = "left" THEN
				x <- x - speed
			IF direction = "down" THEN
				y <- y + speed
			IF direction = "up" THEN
				y <- y - speed
			ENDIF
		ENDFOR

		call checkCollision(originalX, originalY)

		set HTML attribute "facing" to moveDirections[0]
		set HTML attribute "walking" to "true"
	ELSE
		set HTML attribute "walking" to "false"
	ENDIF
ENDSUBROUTINE
```

Entity.attack() pseudocode:

```
SUNBROUTINE attack(target)
	IF HTML attribute "attacking" != "true" THEN
		set HTML attribute "attacking" to "true"

		IF target.x < x THEN
			set HTML attribute "facing" to "left"
		ELSE IF target.x > x THEN
			set HTML attribute "facing" to "right"
		ENDIF

		target.damage(attackDamage)
		WAIT attackCooldown seconds
		set HTML attribute "attacking" to false
	ENDIF
ENDSUBROUTINE
```

Enemy pathfinding and movement

note: refer to the game processes flowchart for a brief overview of the order of the enemies' actions

For the enemies to act as required, and follow the player around the maze, they need some kind of logic system to determine when they should move, how they should move and where they should move to.

The movement of the enemies is split up into two sections:

- moving towards a target in the tile the enemy is currently in
- moving towards a target in a different tile

In the context of this game, the enemies main target will always be the player - however there is one instance where this won't be the case.

First of all, if we assume that any enemy will have the player enter its range (the radius around it that it can path find in) before it can possibly reach the enemy, then at this point, the player won't be in the same tile as the enemy, so we must use the second type of movement.

Before we can do this however, we need to determine the path the enemy needs to take to reach the player - more on this later. If a path exists and the enemy can move to the player, the path will be a list containing directions that the enemy needs to follow (*note: this list is used as a queue - directions are added to the end and read and removed from the start*). The enemy can then take the first direction from the queue and move in that direction, until it reaches a new node, when it needs to start moving in the next direction from the path.

At this point, if the enemy started moving in the next direction straight away, then it could, and probably would, get caught on one of the walls, and not be able to move any further in the direction it wanted to. Now, if we set the centre of the tile to the target position that we want the enemy to move towards, the enemy won't get caught on the wall - this is the first type of movement.

For this movement, we can simply compare the x and y coordinates of the enemy to the x and y coordinates of the position we want it to move to, and then make the enemy move in the appropriate directions until it reaches the specified coordinates. In the case of moving the enemy to the centre of a tile, if we check that a target position exists before attempting to move based on the path, we can leave the path as it is whilst the enemy moves towards the tile's centre, then return to the path once this movement is complete - allowing the enemy to move into the next tile. By repeating this sequence for every tile the enemy enters, it will move smoothly through the maze: if it is moving downwards from one node to another, only the y coordinate will change as it should already be at the correct x coordinate for the centre of the tile; on corners, the enemy will move to the centre of the tile from the track that it is already on, before moving in another direction when it reaches the tile's centre.

When the enemy either reaches a tile containing the player, instead of setting the centre of the tile to the target position, the x and y coordinate of the player can be set - these will need to be checked every frame to account for the player's own movement.

This is the process used to begin to come up with the solution for the enemy's movement, the pseudocode for which is shown below.

Enemy.move() pseudocode:

```
SUBROUTINE move()
        call determineTileOrigin()
        targetTile <- player.currentTile

        IF targetTile = currentTile THEN
                target.x <- player.x
                target.y <- player.y
                path <- []
        ENDIF

        IF target.x != -1 AND target.y != -1 THEN
                moveDirections <- []

                IF x > target.x THEN
                        push "left" to moveDirections
                ELSE IF x < target.x THEN
                        push "right" to moveDirections

                IF y > target.y THEN
                        push "up" to moveDirections
                ELSE IF y < target.y THEN
```

```
                        push "down" to moveDirections

                IF moveDirections contains items THEN
                        call parent.move()
                ELSE
                        target.x <- -1
                        target.y <- -1
                        remove path[0] from path
                ENDIF
        ENDIF

        IF path contains no items THEN
                target.x <- -1
                target.y <- -1
                call pathFind()
        ENDIF

        IF path contains items AND target.x = -1 AND target.y = -1 THEN
                moveDirections <- []
                push path[0] to moveDirections
                call parent.move()
                IF currentTile != previousTile THEN
                        target.x <- tileOrigin.x + 25
                        target.y <- tileOrigin.y + 25
                ENDIF
        ENDIF
ENDSUBROUTINE
```

The next problem to solve is how the enemy will actually go about finding a path from itself to the player – the first piece of useful information I realised was that in most cases, it will be more efficient to find the path from the player to the enemy, then have the enemy move along the reverse of that path. Considering the same case as earlier, when the player first enters the enemy's range, the player will have between 1 and 3 possible routes to search, as one will always leave the enemy's range straight away. However, the enemy will have between 2 and 4 routes to search, as it does not have this constraint.

To actually find the path, we can employ another recursive backtracking algorithm (in this instance, acting as a depth-first search of the nodes within the enemy's range). As a recursive backtracking algorithm was used to generate the maze, a recursive backtracking algorithm is the best algorithm to use for this search - due to the long and winding corridor nature of the mazes, there is unlikely to be more than one route for the player to search that both stays within the enemy's range and is more than a few nodes in length, hence why a depth-first search should be used. (If, for example, Primm's algorithm was used to generate the maze, which would generally have far more branching paths, the use of a breadth-first search should be considered).

Before the enemy can begin a recursive backtracking search from the player to itself, it needs to know:

- where the player is
- what tiles are in its range (it is important that this includes both nodes and edges)

Once it has this information, the enemy can look at the tile the player is in, check which of its adjacent tiles are inside its range, then select one of those as the next tile to check. If there are no available adjacent nodes, the node currently being checked will be removed from the list of available nodes, and the last node checked will be selected to be checked again. This process is repeated until either the enemy is found, in which case there will be a path from the enemy to the player for the enemy to follow, or there are no more available nodes and the enemy has not been found, in which case there is not a path from the enemy to the player within the enemy's range.

Enemy.pathfind() pseudocode:

```
SUBROUTINE pathFind()
	targetPosition <- targetTile.position()

	min <- {y: currentTile.y - range/2, x: currentTile.x - range/2}
	max <- {y: currentTile.y + range/2, x: currentTile.x + range/2}

	IF min.y <= targetPosition.y AND targetPosition.y <= max.y AND min.x <= targetPosition.x AND
targetPosition.x <= max.x THEN
		nodesInRange <- new NodeList object

		FOR row <- min.y to max.y, incrementing row by 0.5 DO
			FOR column <- min.x to max.x, incrementing row by 0.5 DO
				node <- maze.getTile(row, column)
				IF node THEN
					push node to nodesInRange
				ENDIF
			ENDFOR
		ENDFOR

		checkTile <- targetTile
		checkPosition <- targetPosition
		visitedNodes <- new NodeList object

		While true DO
			IF checkPosition = currentTile.position() THEN
				break
			ENDIF

			nodesInRange.delete(checkPosition)
			nextPosition <- checkPosition
			direction <- ""

			IF checkTile.top = 0 AND nodesInRange.contains({y: nextPosition.y - 0.5, x:
nextPosition.x})) THEN
				nextPosition.y <- nextPosition.y - 0.5
				direction <- "down"

			ELSE IF checkTile.bottom = 0 AND nodesInRange.contains({y: nextPosition.y + 0.5,
x: nextPosition.x})) THEN
				nextPosition.y <- nextPosition + 0.5
				direction <- "up"

			ELSE IF checkTile.left = 0 AND nodesInRange.contains({y: nextPosition.y, x:
nextPosition.x - 0.5})) THEN
				nextPosition.x <- nextPosition - 0.5
				direction <- "right"

			ELSE IF checkTile.right = 0 AND nodesInRange.contains({y: nextPosition.y, x:
nextPosition.x + 0.5})) THEN
				nextPosition.x <- nextPosition + 0.5
				direction <- "left"

			ELSE
				IF path contains items THEN
					remove path[0] from path
					checkPosition <- visitedNodes.pop()
				ELSE
					break
				ENDIF
			ENDIF

			IF nodesInRange.contains(nextPosition) THEN
				remove last item from path
				visitedNodes.push(checkPosition)
				checkPosition <- nextPosition
				checkTile <- nodesInRange.dict[nodesInRange.getKeyFromPos(checkPosition)]
			ENDIF
```

```
            ENDWHILE
        ENDIF
ENDSUBROUTINE
```

Calculating the score

Score calculation requirements and how they will be achieved:

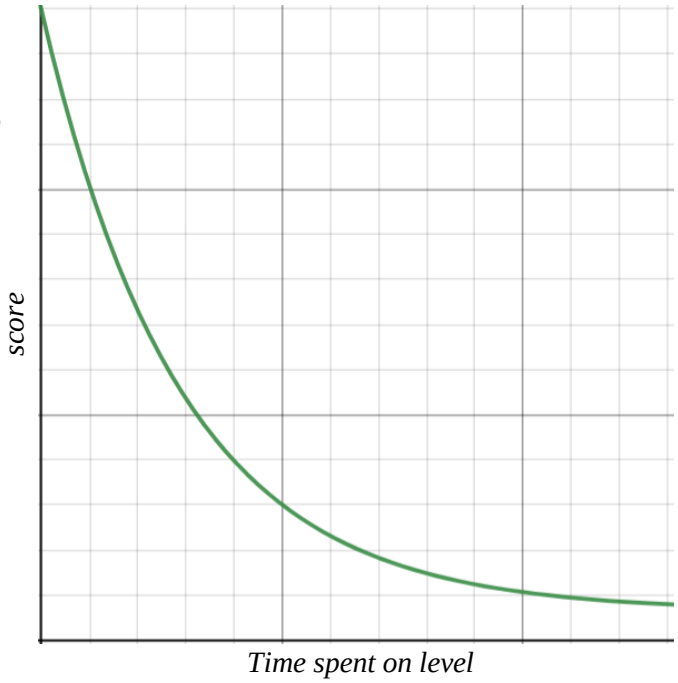
The score decreases as the time spent on the level increases:	The graph of 2^x gives a desirable shape to fulfil this
The score should never be 0 for a completed level	Adding a constant to the graph of 2^x prevents the score from ever reaching 0
The score should take into account the number of enemies defeated and the number of locks opened	Add 10 points for every enemy and 15 for every lock
The score should increase as the size of the maze increases	Multiply the total by the mean of the side lengths

The following equation fits all of the requirements, whilst providing outputs within a reasonable range

$$\frac{\text{height} + \text{width}}{2}\left(2^{-\left(\frac{\textit{time spend on level}}{200} - 10\right)} + 50 + (\textit{enemies killed} \cdot 10) + (\textit{locks opened} \cdot 15)\right)$$

This equation produces a graph of the following shape:

(note: the constant values provided by the number of enemies killed and locks opened translates this curve in the positive y direction, so the shape does not change, but the total score increases)



Testing

Testing evidence can be found at: https://youtu.be/zEM1ttAjdpM

Test 1: The application should be hosted as a website

This group of tests will show that the website can be accessed as a website and using different HTML 5 supporting browsers.

Test Number	Objective being tested	Test data used	Category of test data	Expected outcome	passed/ failed	Timestamp
1.1	2, 2.1	The user should navigate to the local IP address provided by the flask app in google chrome	normal	The home page (`\`) should be displayed	Passed	00:04
1.2	2, 2.1	The user should navigate to the local IP address provided by the flask app in microsoft	normal	The home page (`\`) should be displayed	Passed	00:29

		edge				
1.3	2, 2.1	The user should navigate to the local IP address provided by the flask app in firefox	normal	The home page (`\`) should be displayed	Passed	00:39

Test 2: The application should generate random, procedurally generated mazes.

This group of tests will both show that the program generates random mazes, and will demonstrate the robustness of the URL query inputs for `/play`.

Tests showing mazes being generated will cross-check the name of the background image of the map element of `level.html` with the name of the image located in /mazeimages in the source section of the browser development tools, and show that this in turn is the same as the image stored in /mazeimages in the server. These factors being the same, and the fact that the relevant image will be the only image located in /mazeimages in the server should be sufficient evidence to prove that random mazes are being generated.

Test Number	Objective being tested	Test data used	Category of test data	Expected outcome/s	passed/ failed	Timestamp
2.1	1, 1.2, 1.2.1	URL query for /play with height and width of 10	normal	10x10 Maze image file saved in /mazeimages The same 10x10 Maze image displayed by webpage	Passed	01:11
2.2	1, 1.2, 1.2.1	URL query for /play with height and width of 3	extreme	3x3 Maze image file saved in /mazeimages The same 3x3 Maze image displayed by webpage	Passed	02:25
2.3	1, 1.2, 1.2.1	URL query for /play with height and width of 25	extreme	25x25 Maze image file saved in /mazeimages The same 25x25 Maze image displayed by webpage	Passed	02:57

2.4	1, 1.2, 1.2.1	URL query for /play with height and width of 2	boundary/invalid	3x3 Maze image file saved in /mazeimages The same 3x3 Maze image displayed by webpage	Passed	03:23
2.5	1, 1.2, 1.2.1	URL query for /play with height and width of 26	boundary/invalid	25x25 Maze image file saved in /mazeimages The same 25x25 Maze image displayed by webpage	Passed	03:56
2.6	1, 1.2, 1.2.1	URL query for /play with height and width of 0	invalid	3x3 Maze image file saved in /mazeimages The same 3x3 Maze image displayed by webpage	Passed	04:37
2.7	1, 1.2, 1.2.1	URL query for /play with height and width of 100	invalid	25x25 Maze image file saved in /mazeimages The same 25x25 Maze image displayed by webpage	Passed	05:03
2.8	1, 1.2, 1.2.1	/play with no query	erroneous	3x3 Maze image file saved in /mazeimages The same 3x3 Maze image displayed by webpage	Passed	05:32
2.9	1, 1.2, 1.2.1	URL query for /play with height of 10, no width	erroneous	10x3 Maze image file saved in /mazeimages The same 10x3 Maze image displayed by webpage	Passed	06:02
2.10	1, 1.2, 1.2.1	URL query for /play with width of 10, no	erroneous	3x10 Maze image file saved in /mazeimages	Passed	06:36

		height		The same 3x10 Maze image displayed by webpage		
2.11	1, 1.2, 1.2.1	URL query for /play with height given as non-integer	erroneous	Redirect to `/`, http status 400: Bad request, value error message in console	Passed	07:04
2.12	1, 1.2, 1.2.1	URL query for /play with width given as non-integer	erroneous	Redirect to `/`, http status 400: Bad request, value error message in console	Passed	07:35
2.13	1.1.1, 1.1.1.1	URL query for /play with height of 5 and width of 12	normal	5x12 Maze image file saved in /mazeimages The same 5x12 Maze image displayed by webpage	Passed	07:57
2.14	1.1.1, 1.1.1.1	URL query for .play with height of 17 and width of 8	normal	17x8 Maze image file saved in /mazeimages The same 17x8 Maze image displayed by webpage	Passed	08:29
2.15	1.1.1, 1.1.1.1	URL query for .play with height of 3 and width of 25	extreme	3x25 Maze image file saved in /mazeimages The same 3x25 Maze image displayed by webpage	Passed	08:56
2.16	1.1.1, 1.1.1.1	URL query for .play with height of 25 and width of 3	extreme	25x3 Maze image file saved in /mazeimages The same 25x3 Maze image displayed by webpage	Passed	09:21
2.17	1.1.1,	URL query for	boundary/i	10x25 Maze	Passed	09:55

	1.1.1.1	.play with height of 10 and width of 100	nvalid	image file saved in /mazeimages The same 10x25 Maze image displayed by webpage		
2.18	1.1.1, 1.1.1.1	URL query for .play with height of 100 and width of 10	boundary/invalid	25x10 Maze image file saved in /mazeimages The same 25x10 Maze image displayed by webpage	Passed	10:24
2.19	1.1.1, 1.1.1.1	URL query for .play with height of 10 and width of 0	boundary/invalid	10x3 Maze image file saved in /mazeimages The same 10x3 Maze image displayed by webpage	Passed	10:48
2.20	1.1.1, 1.1.1.1	URL query for .play with height of 0 and width of 10	boundary/invalid	3x10 Maze image file saved in /mazeimages The same 3x10 Maze image displayed by webpage	Passed	11:16
2.21	1.3	Going to /play/[seed] in a new session	normal	Maze with given seed is generated and served to webpage	Passed	11:41
2.22	1.3	Going to /play/	erroneous	404 error page should be displayed	Passed	12:55
2.23	1.3	Going to /play/cheese (invalid seed)	erroneous	Redirect to`/`, http status 500, key error message in console	Passed	13:13
2.24	1.3	Going to /play/= (invalid seed)	erroneous	Redirect to`/`, http status 500, value error message in console	Passed	13:40
2.25	1.3	Changing one	boundary	Redirect to`/`,	Failed	14:02

		of the base 64 values of a seed		http status 500, value error message in console		

Test 3: There should be a series of 10 levels of progressive difficulty for the user to play through

These tests will demonstrate the actual function of the game itself works as intended, by demonstrating specific gameplay examples.

Note: the randomness of the mazes (for objective 3.1.1) is displayed throughout tests, where most tests with normal test data should display an randomly unique maze

Note 2: for tests which do not rely on the presence of enemies, the `maxEnemies` parameter will be set to 0

Test Number	Objective being tested	Test data used	Category of test data	Expected outcome	passed/failed	Timestamp
3.1	3.1., 3.1.1	Pressing the “play” button on the home page	normal	Sent to /play/[seed] with a random 3x3 maze	Passed	15:53
3.2	3.4, 3.4.1	Using W, A, S and D in turn	normal	The player should move up, left, down and then right	Passed	16:05
3.3	3.2, 3.2.2, 3.2.3, 3.4.3.2	Moving through the maze towards the enemy	normal	The enemy should move to the play, and begin dealing damage. The player’s health bar should decrease	Passed	16:36
3.4	3.3, 3.3.1, 3.3.2, 3.3.3 3.5.1, 3.5.1.1, 3.4.3.5,	Picking up, moving and dropping a key, then picking it up again	normal	The key should be added to the player’s inventory and removed from the maze when picked up, then returned to the maze where it is dropped, and should be added to the inventory again	Passed	16:55
3.5	3.4.2	Walking into the walls around the maze	extreme	The player should hit the visual walls, then be stopped from moving past them	Passed	17:18

3.6	3.4.3, 3.4.3.1	In 3x3 maze, game.player. maxHealth variable should be checked in the browser console	extreme	A value of 9 should be returned	Passed	19:57
3.7	3.4.3, 3.4.3.1	In 10x10 maze, game.player. maxHealth variable should be checked in the browser console	extreme	A value of 100 should be returned	Passed	20:15
3.8	3.4.3, 3.4.3.1	In 25x25 maze, game.player. maxHealth variable should be checked in the browser console	extreme	A value of 625 should be returned	Passed	20:29
3.9	3.4.3.3, 3.4.3.3.1	Enemies should be allowed to attack the player until the players health reaches 0 The restart button should be pressed	normal	The game over screen should be displayed, before the user is returned to the home page.	Passed	20:58
3.10	3.4.3.4	The player should attempt to pick up an item when the inventory contains 5 items	extreme	The item should not be picked up, and should remain in the maze	Passed	21:32
3.11	3.4.3.5	The player should use the weapon to attack an enemy	normal	The enemy's health bar should decrease by half on the first attack, and the enemy should be removed	Passed	22:11

				from the maze on the second. The “enemies defeated” count in the game status should increase by one		
3.12	3.5, 3.5.1, 3.5.1.2	The player should attempt to pick up a lock	extreme	The lock should remain in the maze, and should not be added to the players inventory	Passed	22:34
3.13	3.5, 3.5.1, 3.5.1.3	The player should use a key to open a lock	normal	The key should be removed from the player’s inventory The lock should be removed from the maze The “locks remaining” count of the game status should decrease by one	Passed	22:57
3.14	3.5, 3.5.2	In mazes of size 3x3, 10x10 and 25x25, the “locks remaining” count of the game status should be checked	normal	The number of locks should increase as the size of the maze increases	Passed	23:19
3.15	3.5, 3.5.3, 3.5.3.1	The player should unlock all the locks in the maze, then use the “retry” button	normal	The level completed message should be displayed, and the user should be returned to the start of the level upon pressing the button	Passed	23:56
3.16	3.5, 3.5.3, 3.5.3.2	The player should unlock all the locks in the maze, then use the “continue”	normal	The level completed message should be displayed, and the user should be taken to the next	Passed	24:30

		button		level: the “level” number of the game status should increase by one, and the size of the maze should have increased from the previous level		

Test 4: Once all 10 levels are completed, the user should unlock a “Free Play” mode

These tests will demonstrate that the gamecomplete page works as intended, and that no invalid data can be inputted into the custom maze form.

Note: when the values to be submitted for the max locks and/or max enemies is not specifically mentioned, they will be left as the default value of -1

Note 2: when the values to be submitted for the height and width are not specified, a value of 10 should be entered for both

Test Number	Objective being tested	Test data used	Category of test data	Expected outcome	passed /failed	Timestamp
4.1	3, 4, 4.2	The user should complete the final level(level 10) and choose to continue	normal	The user should be redirected to `\gamecomplete`, and the mazes completed in the session should be displayed as links	Passed	25:08
4.2	4.1, 4.1.1, 4.1.2, 4.1.3	The user should enter height and width of 10, and set the max locks and max enemies to 3 each into the custom maze form, before submitting it	normal	The user should be redirected to a “custom” level , with a width and height of 10, with 3 locks remaining and 3 enemies present	Passed	25:58
4.3	4.1, 1.1.1.2	The user should submit the form with a height and width of 3	extreme	The user should be redirected to a custom level with a 3x3 maze	Passed	26:46
4.4	4.1,	The user	boundary	The user should	Passed	27:05

	1.1.1.2	should attempt to submit the form with a height of 2 and a width of 10		be prompted that the minimum value for the height is 3		
4.5	4.1, 1.1.1.2	The user should attempt to submit the form with a width of 2 and a height of 10	boundary	The user should be prompted that the minimum value for the width is 3	Passed	27:16
4.6	4.1, 1.1.1.3	The user should submit the form with a height and width of 25	extreme	The user should be redirected to a custom level with a 25x25 maze	Passed	27:31
4.7	4.1, 1.1.1.3	The user should attempt to submit the form with a height of 26 and a width of 10	boundary/ erroneous	The user should be prompted that the maximum value for the height is 25	Passed	27:46
4.8	4.1, 1.1.1.3	The user should attempt to submit the form with a width of 26 and a height of 10	boundary/ erroneous	The user should be prompted that the maximum value for the width is 25	Passed	28:00
4.9	4.1, 4.1.2, 4.1.3	The user should enter values of 10 for the height and width, and a value of 0 for the max locks and max enemies	extreme	The level the user is redirected to should contain no locks or enemies	Passed	28:14
4.10	4.1, 4.1.2, 4.1.3	The user should enter values of 10 for the height and width, and leave the max locks	normal	The number of locks in the custom maze should be the same as the number of locks in the maze	Passed	28:44

		and max enemies as the default of -1 The user should then take the seed for the maze that is generated, and copy it into a new browser window		generated from the seed The number of enemies in the custom maze should be the same as the number of enemies in the maze generated from the seed		
4.11	4.1, 4.1.2, 4.1.3	The user should enter values 999 for the max locks and max enemies The user should then take the seed for the maze that is generated, and copy it into a new browser window	extreme	The number of locks in the custom maze should be <=999 The number of enemies in the maze should be <=999	Passed	29:44
4.12	4.1, 4.1.2	The user should set the value for max locks as -2	boundary/ erroneous	The user should be prompted that the minimum value for the max locks is -1	Passed	31:24
4.13	4.1, 4.1.2	The user should set the value for max locks as 1000	boundary/ erroneous	The user should be prompted that the maximum value for the max locks is 999	Passed	31:43
4.14	4.1, 4.1.2	The user should set the value for max enemies as -2	boundary/ erroneous	The user should be prompted that the minimum value for the max enemies is -1	Passed	31:59
4.15	4.1, 4.1.2	The user should set the value for max enemies as 1000	boundary/ erroneous	The user should be prompted that the maximum value for the max enemies is 999	Passed	32:09

4.16		The user should complete the first maze in the maze list, the select the “continue” button	normal	The user should be returned to the gamecomplete screen after completing the maze	Passed	32:24
4.17		The user should complete a 3x3 maze generated from the custom maze form, and select the “continue” button	normal	The user should be returned to the gamecomplete screen after completing the maze	Passed	32:54

Test 5: There should be a scoring system for the levels

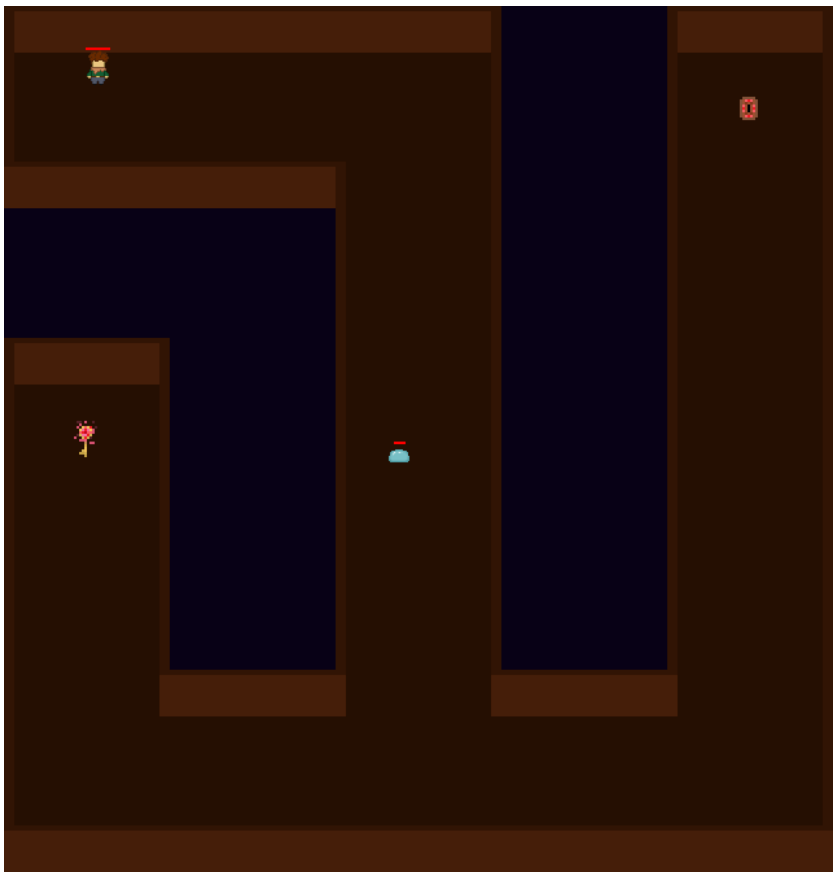
Test 5 will be carried out as an experiment, recording specific data about maze playthroughs which follow specific criteria. Overall, the results of these playthroughs should line up with the expected outcome to confirm that the relevant objectives have been met.

Test Number	Objective being tested	Test data used	Category of test data	Expected outcome	passed/failed	evidence
5	5.1, 5.2	The user should complete the mazes with the conditions specified in the table, and record the scores, enemies defeated, locks opened and time elapsed	normal	When the final lock is opened for each level, the level completed message should be displayed, with a value for the score underneath It should be shown that the score decreases for a lower amount of time spent, increases for a higher amount of locks opened and enemies defeated, and increases with the maze size	Passed	33:40

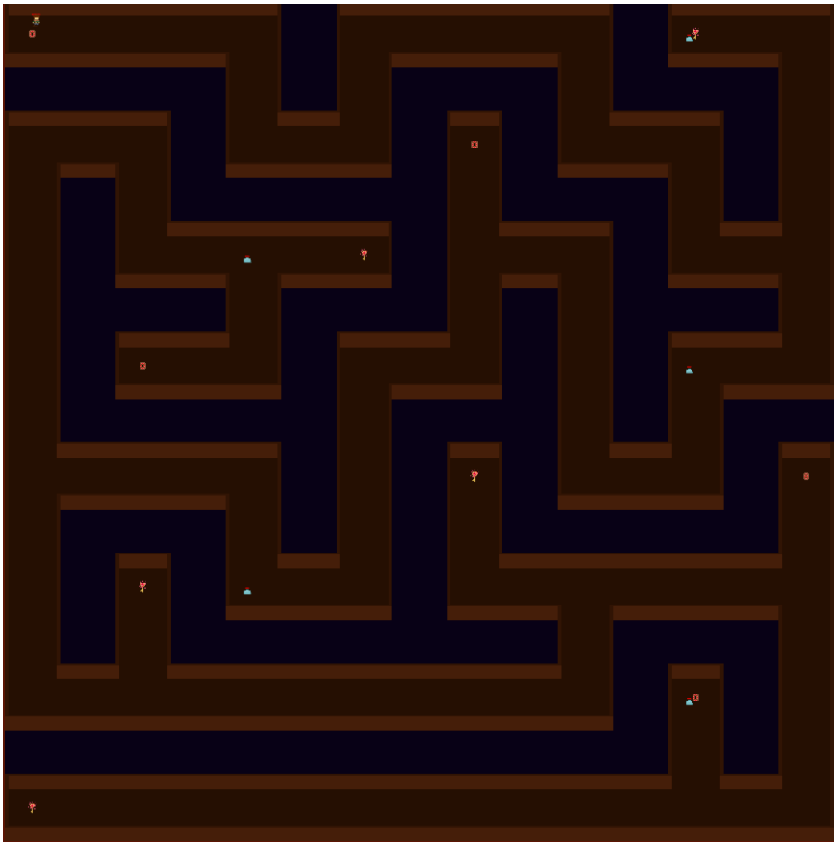
*Note: items in **bold** are those entered by the tester, all other items are to be followed*

Height x width	seed	score	Enemies defeated	Locks opened	Time elapsed	Condition to follow	timestamp
3x3	AwPpuzNkUAAA==	3202	1	1	0:09	Completed as quickly as possible	33:42
3x3	AwPpuzNkUAAA==	3172	0	1	0:09	Completed as quickly as possible, without defeating enemy	33:57
3x3	AwPpuzNkUAAA==	2720	1	1	1:00	Enemy defeated, lock opened after a minute of waiting	34:13
3x3	AwPpuzNkUAAA==(no enemy)	2690	0	1	1:00	No enemy present, lock opened after a minute of waiting	34:37
3x3	AwPpuzNkUAAA==	3127	0	0	0:09	No locks opened or enemies defeated, killed by enemy	35:01
8x8	CAjsmsnpqWW2kzaNKWE-WlOlLJO2WztlaMFkzMWz7MzMRQAA==	6477	5	5	2:17	Completed to a normal pace, all enemies defeated, all locks opened	35:18
8x8	CAjsmsnpqWW2kzaNKWE-WlOlLJO2WztlaMFkzMWz7MzMRQAA==	5938	0	5	2:25	All locks opened, no enemies defeated	36:22
8x8	CAjsmsnpqWW2kzaNKWE-WlOlLJO2WztlaMFkzMWz7MzMRQAA==	7457	0	0	0:43	No locks opened, no enemies defeated, killed by enemy	37:47

Birds-Eye View of the Tested Mazes



AwPpuzNkUAAA==



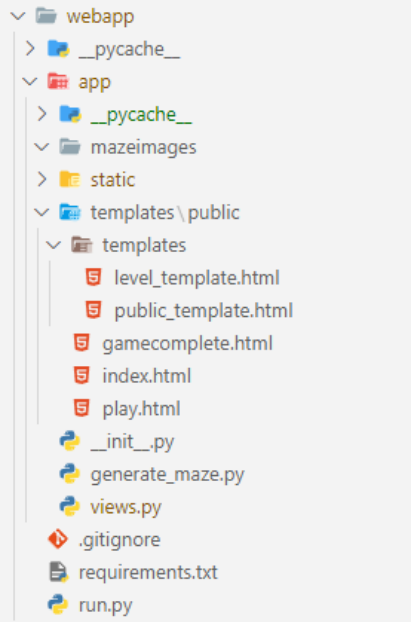
CAjsmsnpqWW2kzaNKWE-WlOlLJO2WztlaMFkzMWz7MzMRQAA==

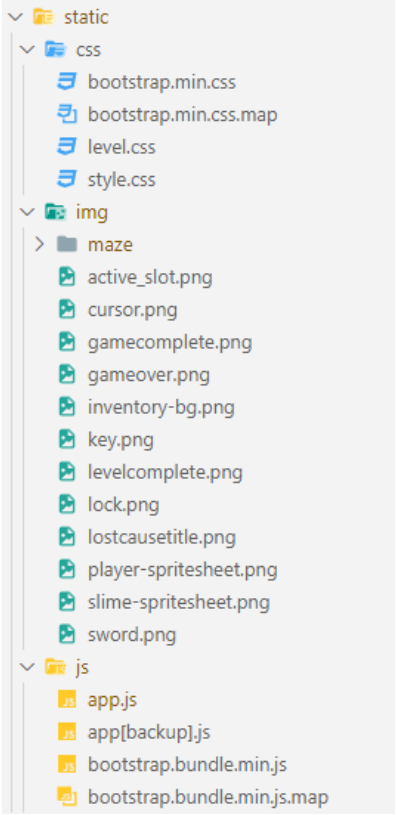
Technical Solution

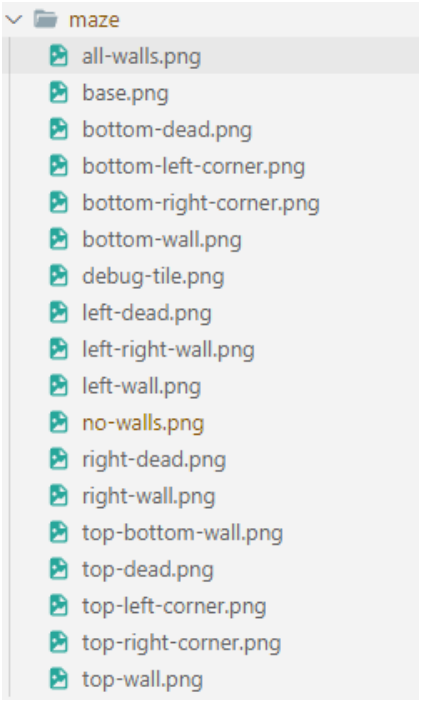
An overview of the file structure of the application.

Folders and Files included in the images which are not essential aspects of the application:

- __pycache__
- .gitignore
- app[backup].js







Starting the app

run.py

```
1. from app import app
2.
3. if __name__ == '__main__':
4.     # host app on default ip
5.     app.run(host='0.0.0.0', port=8080)
```

requirements.txt

```
1. click==8.0.1
2. Flask==2.0.2
3. itsdangerous==2.0.1
4. Jinja2==3.0.2
5. MarkupSafe==2.0.1
```

```
6. Werkzeug==2.0.2
```

__init__.py

```
1. from flask import Flask
2.
3. # initialises app
4. app = Flask(__name__)
5.
6. from app import views
```

Flask Server

views.py

```
1. from app import app, generate_maze
2. from flask import render_template, redirect, send_from_directory, request, session,
   abort
3. import os
4. from time import time
5. from PIL import Image
6.
7. app.secret_key = os.urandom(12).hex()
8. maze_image_dir = os.path.join(app.root_path, "mazeimages")
9. seed_generator = generate_maze.SeedGenerator()
10.
11.
12.def save_maze_image(maze_image):
13.    """
14.    It deletes all the files in the maze_image_dir directory, then saves the
   maze_image parameter as a PNG file in the
15.    maze_image_dir directory
16.
17.    :param maze_image: the image of the maze
18.    """
19.    for file in os.scandir(maze_image_dir):
20.        os.remove(file)
21.    # stores the name of the image currently in use
22.    session['image name'] = f"maze{str(time()).split('.')[0]}.png"
23.    maze_image.save(os.path.join(maze_image_dir, session['image name']))
24.
25.
26.def add_maze_to_session(seed):
27.    """
28.    If the session doesn't have a key called 'maze list', create it and set it to an
   empty list. If the seed isn't already
29.    in the list, add it
30.
31.    :param seed: the seed for the maze
32.    """
33.    if 'maze list' not in session:
34.        session['maze list'] = []
35.    if seed not in session['maze list']:
36.        session['maze list'].append(seed)
37.
38.
```

```
39. def checkSideLength(length):
40.     """
41.     If the length is less than 3, set it to 3. If the length is greater than 25, set
    it to 25. Otherwise, leave it alone.
42.
43.     :param length: The length of the side of the maze
44.     :return: The length of the side of the maze.
45.     """
46.     if length < 3:
47.         length = 3
48.     elif length > 25:
49.         length = 25
50.     return length
51.
52.
53. @app.route('/')
54. def index():
55.     """
56.     The function `index()` returns the rendered template `public/index.html`
57.     :return: The index.html file is being returned.
58.     """
59.     return render_template('public/index.html')
60.
61.
62. @app.route('/mazeimages/<string:file_name>')
63. def get_maze_image(file_name):
64.     """
65.     It returns the image file from the maze_image_dir directory
66.
67.     :param file_name: The name of the file to be sent
68.     :return: The image of the maze.
69.     """
70.     return send_from_directory(maze_image_dir, file_name)
71.
72.
73. @app.route('/play', methods=['GET'])
74. def play_with_size():
75.     """
76.     It takes in the height and width of the maze, creates a maze with those
    dimensions, and then redirects the user to the
77.     play page for that maze
78.     :return: A redirect to the play page with the seed as a parameter.
79.     """
80.
81.     args = request.args.to_dict()
82.     try:
83.         # takes query parameters and stores them either in the session or the seed
    generator
84.         if 'maxEnemies' in args:
85.             session['max enemies'] = int(args['maxEnemies'])
86.         else:
87.             session['max enemies'] = -1
88.
89.         if 'maxLocks' in args:
90.             session['max locks'] = int(args['maxLocks'])
91.         else:
92.             session['max locks'] = -1
93.
94.         if 'height' in args:
95.             seed_generator.height = checkSideLength(int(args['height']))
96.         else:
97.             seed_generator.height = 3
```

```
98.
99.         if 'width' in args:
100.                    seed_generator.width = checkSideLength(int(args['width']))
101.                else:
102.                    seed_generator.width = 3
103.            except ValueError as err:
104.                # called if any of the arguments are invalid
105.                print(f"Bad request | {type(err)} {err}")
106.                return redirect("/", 400)
107.
108.            maze_image = seed_generator.create_base_64_seed()
109.            save_maze_image(maze_image)
110.            add_maze_to_session(seed_generator.seed)
111.
112.            return redirect(f"/play/{seed_generator.seed}")
113.
114.
115.        @app.route('/play/<string:seed>')
116.        def play_from_seed(seed):
117.            """
118.            It loads the maze image and adds its name to the session if it's not already
    there, then renders the play page
119.
120.            :param seed: the seed for the maze
121.            :return: The play.html template is being returned.
122.            """
123.            if seed_generator.seed != seed:
124.                try:
125.                    seed_generator.seed = seed
126.                    maze_image = seed_generator.draw_maze_from_seed()
127.                except (IndexError, KeyError, ValueError, AttributeError) as err:
128.                    # called if the given seed is not valid
129.                    print(f"Invalid seed | {type(err)} {err}")
130.                    return redirect("/", 500)
131.
132.                save_maze_image(maze_image)
133.                add_maze_to_session(seed)
134.
135.            if 'game complete' not in session:
136.                session['game complete'] = 0
137.
138.            if session['game complete']:
139.                level = "custom"
140.            else:
141.                level = str(len(session['maze list']))
142.
143.            # do not want to keep any values set for max enemies or max locks between
    mazes
144.            if 'max enemies' in session:
145.                enemies = session['max enemies']
146.                del session['max enemies']
147.            else:
148.                enemies = -1
149.
150.            if 'max locks' in session:
151.                locks = session['max locks']
152.                del session['max locks']
153.            else:
154.                locks = -1
155.
```

```
156.          return render_template('public/play.html', level=level, maxEnemies=enemies,
   maxLocks=locks, mazeImage=session['image name'], mazeSeed=seed_generator.seed,
   gameComplete=session['game complete'])
157.
158.
159.      @app.route('/gamecomplete')
160.      def game_complete():
161.          """
162.          This function is called when the user completes the game. It sets the
   session variable 'game complete' to 1, and then
163.          renders the gamecomplete.html template.
164.          :return: The gamecomplete.html page is being returned.
165.          """
166.
167.          if 'maze list' not in session:
168.              abort(401)
169.
170.          session['game complete'] = 1
171.
172.          return render_template('public/gamecomplete.html', mazeList=session['maze
   list'])
```

generate_maze.py

```
1.  from dataclasses import dataclass
2.  import random
3.  import time
4.  from PIL import Image
5.  from pathlib import Path
6.
7.
8.  @dataclass
9.  class Node:
10.     __pos: list
11.
12.     def __post_init__(self):
13.         """
14.         The function takes the position of the cell and assigns it to the row and
   column variables
15.         """
16.         self._top = self._bottom = self._left = self._right = "1"
17.
18.         self._row = self.__pos[0]
19.         self._column = self.__pos[1]
20.
21.     def _wall_value_checker(self, value):
22.         """
23.         It checks if the value of a wall is either "1" or "0"
24.
25.         :param value: the value of the cell
26.         :return: a boolean value.
27.         """
28.         if value == "1" or value == "0":
29.             return True
30.         else:
31.             raise ValueError("wall value should be either \"1\" or \"0\"")
32.
33.     def debug(self):
34.         """
```

```
35.         It returns a list of the position of the node, the node's walls, and the
    object's key
36.         :return: The position of the node, the boundaries of the node, and the key of
    the node.
37.         """
38.         return [self.__pos, (self._top, self._bottom, self._left, self._right),
    self.key]
39.
40.     def setWallsFromKey(self, key):
41.         self.top = key[0]
42.         self.bottom = key[1]
43.         self.left = key[2]
44.         self.right = key[3]
45.
46.     @property
47.     def top(self):
48.         """
49.         getter for property "top"
50.         :return: the value of the top wall
51.         """
52.         return self._top
53.
54.     @top.setter
55.     def top(self, value):
56.         """
57.         This function checks if the value is a valid wall value, and if it is, it sets
    the top wall to that value
58.
59.         :param value: The value of the wall
60.         """
61.         if self._wall_value_checker(value):
62.             self._top = value
63.
64.     @property
65.     def bottom(self):
66.         """
67.         getter for property "bottom"
68.         :return: The value for the bottom wall
69.         """
70.         return self._bottom
71.
72.     @bottom.setter
73.     def bottom(self, value):
74.         """
75.         It checks if the value is a valid wall value and if it is, it sets the bottom
    wall to that value.
76.
77.         :param value: The value of the wall
78.         """
79.         if self._wall_value_checker(value):
80.             self._bottom = value
81.
82.     @property
83.     def left(self):
84.         """
85.         getter for property "left"
86.         :return: The value for the left wall
87.         """
88.         return self._left
89.
90.     @left.setter
91.     def left(self, value):
```

```
92.         """
93.         This function takes in a value and checks if it is a valid wall value. If it
    is, it sets the left wall value to the
94.         value
95.
96.         :param value: The value of the wall
97.         """
98.         if self._wall_value_checker(value):
99.             self._left = value
100.
101.          @property
102.          def right(self):
103.              """
104.              getter for property "right"
105.              :return: The value for the right wall
106.              """
107.              return self._right
108.
109.          @right.setter
110.          def right(self, value):
111.              """
112.              This function checks if the value is a valid wall value, and if it is,
    it sets the right wall to that value
113.
114.              :param value: The value of the wall
115.              """
116.              if self._wall_value_checker(value):
117.                  self._right = value
118.
119.          @property
120.          def row(self):
121.              """
122.              getter for property "row"
123.              :return: The row the node is in
124.              """
125.              return self._row
126.
127.          @property
128.          def column(self):
129.              """
130.              getter for property "column"
131.              :return: The column the node is in
132.              """
133.              return self._column
134.
135.          @property
136.          def key(self):
137.              """
138.              The key for each node is a 4 digit binary string, where the first two
    digits are the top and bottom edges, and the
139.              last two digits are the left and right edges.
140.              :return: A 4 digit binary string.
141.              """
142.              return self._top + self._bottom + self._left + self._right
143.
144.
145.      class Maze:
146.          def __init__(self, max_y, max_x):
147.              """
148.              It creates a 2D list of node objects, where the index of the list
    matches the coordinate in the maze
149.
```

```
150.                :param max_y: The height of the maze
151.                :param max_x: The width of the maze
152.                """
153.                self._max_x = max_x
154.                self._max_y = max_y
155.                self._node_list = [[None for _ in range(max_x)] for _ in range(max_y)]
156.                self._binary_list = ['' for _ in range(max_y * max_x)]
157.
158.            def insert(self, node_object):
159.                """
160.                The function takes a node object and inserts it into the node list and
    binary list
161.
162.                :param node_object: the node object to be inserted into the maze
163.                """
164.                row = node_object.row
165.                column = node_object.column
166.                # maze starts at 1,1 but list indexing starts at [0][0]
167.                self._node_list[row - 1][column - 1] = node_object
168.                # calculates relative position
169.                self._binary_list[(self._max_x * (row - 1)) + (column - 1)] =
    f'{node_object.top}{node_object.bottom}{node_object.left}{node_object.right}'
170.
171.            def node(self, coord):
172.                """
173.                It returns the node at the given coordinate
174.
175.                :param coord: The coordinate of the node you want to get
176.                :return: The node at the given coordinates.
177.                """
178.                node = self._node_list[coord[0] - 1][coord[1] - 1]
179.                return node
180.
181.            @property
182.            def binary_string(self):
183.                """
184.                It takes a list of strings, and joins them together into one string
185.                :return: The binary string is being returned.
186.                """
187.                return ''.join(self._binary_list)
188.
189.
190.        class MazeGenerator:
191.            def __init__(self, max_y, max_x):
192.                """
193.                The function takes in the dimensions of the maze and creates a 2D list
    of Nodes
194.
195.                :param max_y: the number of rows in the maze
196.                :param max_x: the number of columns in the maze
197.                """
198.                self._maze = Maze(max_y, max_x)
199.
200.                self._nodes = []
201.
202.                # nodes is a 2D list, [row][column]
203.                for row in range(1, max_y + 1):
204.                    for column in range(1, max_x + 1):
205.                        self._nodes.append([row, column])
206.
207.                # allows easy calculation of the nodes around any given node
208.                self._adjacent_coords = ((-1, 0), (1, 0), (0, 1), (0, -1))
```

```
209.
210.             self._max_x = max_x
211.             self._max_y = max_y
212.
213.             # number corresponds to binary value of the node's key
214.             self._tile_names = {'0000': 'no-walls.png', '0001': 'right-wall.png',
    '0010': 'left-wall.png',
215.                                 '0011': 'left-right-wall.png',
216.                                 '0100': 'bottom-wall.png', '0101': 'bottom-right-
    corner.png',
217.                                 '0110': 'bottom-left-corner.png',
218.                                 '0111': 'bottom-dead.png', '1000': 'top-wall.png',
219.                                 '1001': 'top-right-corner.png', '1010': 'top-left-
    corner.png', '1011': 'top-dead.png',
220.                                 '1100': 'top-bottom-wall.png', '1101': 'right-
    dead.png',
221.                                 '1110': 'left-dead.png', '1111': 'all-walls.png'}
222.
223.         def draw_maze(self, binary_string):
224.             """
225.             It takes a binary string, and uses it to generate a maze image
226.
227.             :param binary_string: the binary string of the maze
228.             :return: The image of the maze.
229.             """
230.
231.             maze_path = Path('app/static/img/maze/')
232.
233.             base = maze_path / 'base.png'
234.             img = Image.open(base)
235.
236.             # tiles are 32x32
237.             img = img.resize((self._max_x * 64 - 32, self._max_y * 64 - 25))
238.
239.             # debug = Image.open(maze_path / 'debug-tile.png')
240.
241.             for row in range(0, self._max_y):
242.                 for column in range(0, self._max_x):
243.
244.                     key = binary_string[:4]
245.                     binary_string = binary_string[4:]
246.
247.                     # pasting the correct image (corresponding with the walls list)
    onto the main background image
248.                     tile = maze_path / self._tile_names[key]
249.                     tile_image = Image.open(tile)
250.                     img.paste(tile_image, (column * 64, row * 64))
251.
252.                     # only needs to check bottom and right or an edge
253.                     if key[1] == "0":
254.                         tile = maze_path / 'left-right-wall.png'  # vertical
    corridor
255.                         tile_image = Image.open(tile)
256.                         img.paste(tile_image, (column * 64, (row * 64) + 32))
257.
258.                     if key[3] == "0":
259.                         tile = maze_path / 'top-bottom-wall.png'  # horizontal
    corridor
260.                         tile_image = Image.open(tile)
261.                         img.paste(tile_image, ((column * 64) + 32, row * 64))
262.
263.             return img
```

```
264.
265.         def recursive_backtracking(self):
266.             """
267.             The function starts at a random node, and then checks all adjacent nodes
    to see if they have been visited. If they
268.             have not, it chooses one at random and removes the wall between the two
    nodes. If they have been visited, it
269.             backtracks to the last node that had an unvisited adjacent node
270.             :return: A maze object
271.             """
272.
273.             stack = []
274.             start_time = time.time()
275.             next_position = [1, 1]
276.
277.             while True:
278.
279.                 current_position = next_position
280.                 stack.append(current_position)
281.
282.                 try:
283.                     self._nodes.remove(current_position)
284.                 except ValueError:
285.                     # value error is raised if the current position has already been
    removed
286.                     # occurs after maze reaches a dead end and backtracks
287.                     pass
288.
289.                 possible_coordinates = []
290.
291.                 # finding possible next nodes by comparing each position adjacent to
    the current node to the unused nodes
292.                 for dy, dx in self._adjacent_coords:
293.                     if [current_position[0] + dy, current_position[1] + dx] in
    self._nodes:
294.                         possible_coordinates.append(
295.                             [current_position[0] + dy, current_position[1] + dx])
296.                 if possible_coordinates:
297.                     # choosing a random (adjacent) node to go next
298.                     next_position = random.choice(possible_coordinates)
299.                     pair = [next_position, current_position]
300.
301.                     # updating (or adding if not already) nodes in maze
302.                     for index, coord in enumerate(pair):
303.                         if not self._maze.node(coord):
304.                             node = Node(coord)
305.                         else:
306.                             node = self._maze.node(coord)
307.
308.                         # working out which wall to remove
309.                         adjacent_node = pair[index - 1]
310.                         y_difference = coord[0] - adjacent_node[0]
311.                         x_difference = coord[1] - adjacent_node[1]
312.                         if y_difference > 0:
313.                             node.top = '0'
314.                         elif y_difference < 0:
315.                             node.bottom = '0'
316.                         elif x_difference > 0:
317.                             node.left = '0'
318.                         elif x_difference < 0:
319.                             node.right = '0'
320.
```

```
321.                         self._maze.insert(node)
322.
323.                 else:
324.                     # checking each node from the stack for possible nodes, if there
    are none, removing it
325.                     for index in range(len(stack) - 1, -1, -1):
326.                         check_position = stack[index]
327.                         for dy, dx in self._adjacent_coords:
328.                             if [check_position[0] + dy, check_position[1] + dx] in
    self._nodes:
329.                                 next_position = check_position
330.                                 break
331.                         else:
332.                             try:
333.                                 # using stack to keep track of which nodes to/ not
    to visit again
334.                                 stack.pop()
335.                             except ValueError:
336.                                 pass
337.                             continue
338.                         break
339.
340.                 if len(stack) == 0:
341.                     break
342.
343.             # end_time = time.time()-start_time
344.             # print((str(end_time)[:-(len(str(end_time).split('.')[1])-2)]) + 's')
345.
346.             return self._maze
347.
348.
349.     class SeedGenerator:
350.         def __init__(self):
351.             """
352.             Defines variables to be used by SeedGenerator
353.             """
354.             self._mazeGenerator = None
355.             self._seed = None
356.             self._maze = None
357.             self._height = None
358.             self._width = None
359.
360.             # base 64 conversion tables
361.             self._toBinary = {'A': '000000', 'B': '000001', 'C': '000010', 'D':
    '000011', 'E': '000100', 'F': '000101',
362.                               'G': '000110', 'H': '000111',
363.                               'I': '001000', 'J': '001001', 'K': '001010', 'L':
    '001011', 'M': '001100', 'N': '001101',
364.                               'O': '001110', 'P': '001111',
365.                               'Q': '010000', 'R': '010001', 'S': '010010', 'T':
    '010011', 'U': '010100', 'V': '010101',
366.                               'W': '010110', 'X': '010111',
367.                               'Y': '011000', 'Z': '011001', 'a': '011010', 'b':
    '011011', 'c': '011100', 'd': '011101',
368.                               'e': '011110', 'f': '011111',
369.                               'g': '100000', 'h': '100001', 'i': '100010', 'j':
    '100011', 'k': '100100', 'l': '100101',
370.                               'm': '100110', 'n': '100111',
371.                               'o': '101000', 'p': '101001', 'q': '101010', 'r':
    '101011', 's': '101100', 't': '101101',
372.                               'u': '101110', 'v': '101111',
```

```
                                'w': '110000', 'x': '110001', 'y': '110010', 'z':
    '110011', '0': '110100', '1': '110101',
                                '2': '110110', '3': '110111',
                                '4': '111000', '5': '111001', '6': '111010', '7':
    '111011', '8': '111100', '9': '111101',
                                '-': '111110', '_': '111111'}

            self._to64 = {'000000': 'A', '000001': 'B', '000010': 'C', '000011':
    'D', '000100': 'E', '000101': 'F',
                          '000110': 'G',
                          '000111': 'H', '001000': 'I', '001001': 'J', '001010':
    'K', '001011': 'L', '001100': 'M',
                          '001101': 'N',
                          '001110': 'O', '001111': 'P', '010000': 'Q', '010001':
    'R', '010010': 'S', '010011': 'T',
                          '010100': 'U',
                          '010101': 'V', '010110': 'W', '010111': 'X', '011000':
    'Y', '011001': 'Z', '011010': 'a',
                          '011011': 'b',
                          '011100': 'c', '011101': 'd', '011110': 'e', '011111':
    'f', '100000': 'g', '100001': 'h',
                          '100010': 'i',
                          '100011': 'j', '100100': 'k', '100101': 'l', '100110':
    'm', '100111': 'n', '101000': 'o',
                          '101001': 'p',
                          '101010': 'q', '101011': 'r', '101100': 's', '101101':
    't', '101110': 'u', '101111': 'v',
                          '110000': 'w',
                          '110001': 'x', '110010': 'y', '110011': 'z', '110100':
    '0', '110101': '1', '110110': '2',
                          '110111': '3',
                          '111000': '4', '111001': '5', '111010': '6', '111011':
    '7', '111100': '8', '111101': '9',
                          '111110': '-',
                          '111111': '_'}

        @property
        def seed(self):
            """
            getter for seed of the maze.
            :return: The seed is being returned.
            """
            return self._seed

        @seed.setter
        def seed(self, seed):
            """
            The function takes in a seed and sets the seed for the maze

            :param seed: The seed for the maze
            """
            self._seed = seed

        @property
        def height(self):
            """
            getter for height of the maze.
            :return: The height of the maze
            """
            return self.height

        @height.setter
```

```
424.          def height(self, height):
425.              """
426.              If the height is less than 10, add a 0 to the front of the height
427.
428.              :param height: The height of the maze
429.              """
430.              if len(str(height)) < 2:
431.                  self._height = int('0' + str(height))
432.              else:
433.                  self._height = height
434.
435.          @property
436.          def width(self):
437.              """
438.              getter for the width of the maze.
439.              :return: The width of the maze
440.              """
441.              return self.width
442.
443.          @width.setter
444.          def width(self, width):
445.              """
446.              If the width less than 10, add a 0 to the front of the width
447.
448.              :param width: The width of the maze
449.              """
450.              if len(str(width)) < 2:
451.                  self._width = int('0' + str(width))
452.              else:
453.                  self._width = width
454.
455.          def create_base_64_seed(self):
456.              """
457.              It takes the maze's width and height, turns them into two bytes, adds
    them to the maze's binary string, pads the
458.              binary string with zeros until it's a multiple of 24, splits the binary
    string into 6 character chunks, and then
459.              converts each chunk into a base 64 character
460.              :return: The image of the maze is being returned.
461.              """
462.              self._mazeGenerator = MazeGenerator(self._height, self._width)
463.
464.              self._maze = self._mazeGenerator.recursive_backtracking()
465.              image = self._mazeGenerator.draw_maze(self._maze.binary_string)
466.
467.              # turns the width and height of the maze into two bytes
468.              size_binary = str(bin(self._height))[2:].zfill(8) +
    str(bin(self._width))[2:].zfill(8)
469.              binary_string = size_binary + self._maze.binary_string
470.
471.              padding = 0
472.              # one '=' is added for every byte of padding
473.              while len(binary_string) % 24 != 0:
474.                  binary_string += '0'
475.                  padding += 1
476.              padding //= 4
477.
478.              # splits binary_string every 6 characters for easier conversion to base
    64
479.              binary_list = [binary_string[i:i + 6] for i in range(0,
    len(binary_string), 6)]
480.
```

```
481.                base_64_string = ''
482.                for value in binary_list:
483.                    base_64_string += self._to64[value]
484.                base_64_string += '=' * padding
485.
486.                self._seed = base_64_string
487.                return image
488.
489.            def draw_maze_from_seed(self):
490.                """
491.                It takes a base64 encoded string, converts it to binary, and then uses
    the binary string to draw a maze
492.
493.                :return: The image for the maze is being returned.
494.                """
495.                # removing padding before conversion
496.                base_64_string = self._seed.rstrip('=')
497.                binary_string = ''
498.
499.                for value in base_64_string:
500.                    binary_string += self._toBinary[value]
501.
502.                # splitting converted string into height, width, and binary_string
503.                padding = self._seed.count('=')
504.                self._height = int(binary_string[:8], 2)
505.                self._width = int(binary_string[8:16], 2)
506.                if padding:
507.                    binary_string = binary_string[16:-padding * 8]
508.                else:
509.                    binary_string = binary_string[16:]
510.
511.                self._mazeGenerator = MazeGenerator(self._height, self._width)
512.
513.                image = self._mazeGenerator.draw_maze(binary_string)
514.
515.                return image
```

Client Side

app.js

```
1. // debug
2. console.log(mazeSeed);
3. const mazeScale = 128;
4. // defining the first instance of pixelSize for it to be used throughout the program
5. let pixelSize =
   parseInt(getComputedStyle(document.documentElement).getPropertyValue('--pixel-size'));
6.
7. // mapping keys to movement directions
8. const directions = {
9.     up: "up",
10.    down: "down",
11.    left: "left",
12.    right: "right"
13.}
14.
15.const directionKeys = {
16.    'w': directions.up,
```

```
    'a': directions.left,
    'd': directions.right,
    's': directions.down,
    'W': directions.up,
    'A': directions.left,
    'S': directions.down,
    'D': directions.right
}

// mapping keys to inventory slots
const inventoryKeys = {
    '1': 1,
    '2': 2,
    '3': 3,
    '4': 4,
    '5': 5
}

// mapping keys to commands
const commands = {
    'e': 'interact',
    'E': 'interact',
    'q': 'drop',
    'Q': 'drop',
    'r': 'attack',
    'R': 'attack'
}

// base 64 to binary conversion table
const toBinary = {'A': '000000', 'B': '000001', 'C': '000010', 'D': '000011', 'E': '000100', 'F': '000101',
                  'G': '000110', 'H': '000111',
                  'I': '001000', 'J': '001001', 'K': '001010', 'L': '001011', 'M': '001100', 'N': '001101',
                  'O': '001110', 'P': '001111',
                  'Q': '010000', 'R': '010001', 'S': '010010', 'T': '010011', 'U': '010100', 'V': '010101',
                  'W': '010110', 'X': '010111',
                  'Y': '011000', 'Z': '011001', 'a': '011010', 'b': '011011', 'c': '011100', 'd': '011101',
                  'e': '011110', 'f': '011111',
                  'g': '100000', 'h': '100001', 'i': '100010', 'j': '100011', 'k': '100100', 'l': '100101',
                  'm': '100110', 'n': '100111',
                  'o': '101000', 'p': '101001', 'q': '101010', 'r': '101011', 's': '101100', 't': '101101',
                  'u': '101110', 'v': '101111',
                  'w': '110000', 'x': '110001', 'y': '110010', 'z': '110011', '0': '110100', '1': '110101',
                  '2': '110110', '3': '110111',
                  '4': '111000', '5': '111001', '6': '111010', '7': '111011', '8': '111100', '9': '111101',
                  '-': '111110', '_': '111111'}


/*  A class that holds a list of objects of the same type */
class ObjectGroup{
    /**
     * The constructor function takes an objectType as an argument and sets the objectType property of the object to the
     * objectType argument. It also sets the objectList property of the object to an empty array
```

```
69.      * @param objectType - This is the type of object that the list will contain.
70.      */
71.     constructor(objectType) {
72.         this.objectType = objectType;
73.         this.objectList = [];
74.     }
75.
76.     /**
77.      * This function returns the object at the specified index in the objectList array
78.      * @param index - The index of the object you want to get.
79.      * @returns The object at the index of the objectList array.
80.      */
81.     get(index){
82.         return this.objectList[index];
83.     }
84.
85.     /**
86.      *  Adds an object to the object list and returns a reference to it
87.      * @param object - The object to be added to the group.
88.      * @returns A reference to the object that was just added to the object list.
89.      */
90.     push(object){
91.         this.objectList.push(object);
92.         return new ObjectGroupReference(object.type, this.objectList.length-1, this);
93.     }
94. }
95.
96.
97. /* A reference to an object in an object group */
98. class ObjectGroupReference{
99.     /**
100.           * This function is a constructor for the object type
    "ObjectPropertiesSection"
101.           * @param objectType - The type of the object.
102.           * @param index - The index of the object in the object group.
103.           * @param objectGroup - The name of the object group that the object belongs
    to.
104.           */
105.          constructor(objectType, index, objectGroup) {
106.              this.objectType = objectType;
107.              this._index = index;
108.              this._objectGroup = objectGroup;
109.          }
110.
111.          /**
112.           * It returns the object at the current index.
113.           * @returns The object at the current index.
114.           */
115.          getSelf(){
116.              return this._objectGroup.get(this._index);
117.          }
118.      }
119.
120.
121.      /* It's a class that represents an item in the game */
122.      class Item{
123.          /**
124.           * It creates a new item entity with the type and id of the item
125.           * @param type - The type of the item.
126.           * @param id - The id of the item.
127.           */
128.          constructor(type, id) {
```

```
129.                 this.type = type;
130.                 this.entity = new ItemEntity(0, 0);
131.                 this.id = id;
132.                 this.entity.self.id = type;
133.             }
134.
135.             /**
136.              * The function place() takes two arguments, y and x, and moves the entity
    to the y and x coordinates
137.              * @param y - The y coordinate of the tile to move to.
138.              * @param x - The x coordinate of the tile to move to.
139.              */
140.             place(y, x){
141.                 this.entity.move(y, x);
142.             }
143.
144.             /**
145.              * When the player picks up the item, the item is removed from the game.
146.              */
147.             take(){
148.                 this.entity.remove();
149.             }
150.
151.             /**
152.              * The update function is called every frame and updates the position of the
    entity
153.              */
154.             update(){
155.                 this.entity.updatePosition();
156.             }
157.         }
158.
159.
160.         /* The Lock class is a subclass of the Item class, and it has a locked property
    that is set to true by default. */
161.         class Lock extends Item{
162.             /**
163.              * The constructor function calls the constructor of the parent to store thr
    type and create an ItemEntity
164.              */
165.             constructor() {
166.                 super('lock')
167.                 this.locked = true;
168.             }
169.
170.             /**
171.              * When the user opens the lock, the lock disappears.
172.              */
173.             unlock(){
174.                 this.locked = false;
175.                 this.entity.self.outerHTML = "";
176.             }
177.         }
178.
179.
180.         /* It's a dictionary that stores nodes, and it also stores the keys in a list */
181.         class NodeList{
182.             /**
183.              * The constructor function creates a new object, and sets the object's dict
    property to an empty object, and sets the
184.              * object's _keys property to an empty array.
185.              */
```

```
186.          constructor() {
187.              this.dict = {};
188.              this._positions = [];
189.          }
190.
191.          /**
192.           * It takes a position object and returns a string that represents the
    position
193.           * @param position - The position of the tile.
194.           * @returns A string that is the key for the position object.
195.           */
196.          getKeyFromPos(position){
197.              return 'y' + String(position.y) + 'x' + String(position.x);
198.          }
199.
200.          /**
201.           * It pushes a node into the grid.
202.           * @param node - The node to be added to the open list.
203.           */
204.          push(node){
205.              let position = node.position();
206.              this.dict[this.getKeyFromPos(position)] = node;
207.              this._positions.unshift(node.position());
208.          }
209.
210.          /**
211.           * If there are keys in the dictionary, remove the first key from the list
    of keys, delete the key from the dictionary,
212.           * and return the position of the key
213.           * @returns The position of the first element in the array.
214.           */
215.          pop(){
216.              if (this._positions){
217.                  let position = this._positions.shift();
218.                  let key = this.getKeyFromPos(position);
219.                  delete this.dict[key];
220.                  return position;
221.              } else {
222.                  return undefined;
223.              }
224.          }
225.
226.          /**
227.           * It deletes the position from the dictionary and the list of positions
228.           * @param position - The position of the object to be deleted.
229.           * @returns The value of the key at the position in the array.
230.           */
231.          delete(position){
232.              let key = this.getKeyFromPos(position);
233.              delete this.dict[key];
234.              this._positions = this._positions.filter(function(e) { return e !==
    position});
235.          }
236.
237.          /**
238.           * It checks if the position is in the dictionary.
239.           * @param position - The position of the tile you want to check.
240.           * @returns A boolean value.
241.           */
242.          contains(position){
243.              let key = this.getKeyFromPos(position)
244.              return key in this.dict;
```

```
245.          }
246.
247.          /**
248.           * If the position is in the maze, return the node at that position
249.           * @param position - The position of the node you want to get.
250.           * @returns The node at the given position.
251.           */
252.          getNode(position){
253.              if (this.contains(position)){
254.                  return this.dict[this.getKeyFromPos(position)]
255.              }
256.              return undefined
257.          }
258.
259.      }
260.
261.
262.      /* A Node is a square that has a position and walls */
263.      class Node{
264.          /**
265.           * The constructor function creates a node object with the given x and y
    coordinates, and the given walls
266.           * @param y - The y coordinate of the node
267.           * @param x - The x coordinate of the node
268.           * @param walls - an object with the following properties:
269.           */
270.          constructor(y, x, walls) {
271.              this.type = 'node'
272.              this.x = x;
273.              this.y = y;
274.              this.top = walls.top;
275.              this.bottom = walls.bottom;
276.              this.left = walls.left;
277.              this.right = walls.right;
278.              this.contains = undefined;
279.          }
280.
281.          /**
282.           * It returns an object with the properties y and x, which are set to the
    values of this.y and this.x
283.           * @returns The object {y: this.y, x: this.x} is being returned.
284.           */
285.          position(){
286.              return {y: this.y, x: this.x};
287.          }
288.      }
289.
290.
291.      /* A horizontal edge is a node with walls on the top and bottom, but not on the
    left or right. */
292.      class HorizontalEdge extends Node{
293.          /**
294.           * The constructor function for the Edge class takes two arguments, y and x,
    and sets the wall properties to
295.           * pre-determined values
296.           * @param y - The y coordinate of the cell
297.           * @param x - The x coordinate of the cell
298.           */
299.          constructor(y, x) {
300.              let walls = { top: 1, bottom: 1, left: 0, right: 0 };
301.              super(y, x, walls);
302.              this.type = 'edge';
```

```
303.          }
304.      }
305.
306.
307.      /* VerticalEdge is a Node with a left and right wall. */
308.      class VerticalEdge extends Node{
309.          /**
310.           * The constructor function for the Edge class takes two arguments, y and x,
   and sets the wall properties to
311.           * pre-determined values
312.           * @param y - The y coordinate of the cell
313.           * @param x - The x coordinate of the cell
314.           */
315.          constructor(y, x) {
316.              let walls = { top: 0, bottom: 0, left: 1, right: 1 };
317.              super(y, x, walls);
318.              this.type = 'edge';
319.          }
320.      }
321.
322.
323.      /* A 2D array of tiles */
324.      class Maze {
325.
326.          /**
327.           * It takes a base64 string, converts it to binary, and then uses the first
   16 bits to determine the height and width
328.           * of the maze
329.           * @param seed - the seed string that is used to generate the maze
330.           */
331.          constructor(seed) {
332.              this.seed = seed;
333.              this._usedEdges = {};
334.              this.nodes = new NodeList();
335.
336.              // removes padding from the seed, so it can be converted into binary
   smoothly
337.              let base64String = this.seed.replace(/=/g, '');
338.              let binaryString = '';
339.              // taking each base 64 character and matching it with the corresponding
   6 binary digits
340.              for (let i = 0; i < base64String.length; i++){
341.                  binaryString += toBinary[base64String[i]];
342.              }
343.
344.              // works out how much padding was used in the seed
345.              let padding = (this.seed.length - base64String.length)*4;
346.
347.              // unpacking the height and width from the front of the binary string,
   and converting back into decimal
348.              this.height = parseInt(binaryString.slice(0,8), 2);
349.              this.width = parseInt(binaryString.slice(8,16), 2);
350.
351.              // removing the height, width and padding from the binary string
352.              this.binaryString = binaryString.slice(16, binaryString.length -
   padding);
353.              console.log(binaryString,this.height, this.width);
354.          }
355.
356.
357.          /**
```

```
358.          * If the coordinates are in the maze, or are within the bounds of the maze
   and are a valid edge, returns true
359.          * if the coordinates are outside the bounds of the maze, or are not a valid
   edge, returns false
360.          * @param y - the y coordinate of the tile
361.          * @param x - the x coordinate of the tile
362.          * @returns A boolean value.
363.          */
364.         checkTileInMaze(y, x) {
365.             if (this.nodes.contains({y: y, x: x})) {
366.                 return true;
367.             } else if ((1 > x || x > this.width) || (1 > y || y > this.height)){
368.                 // if the coordinates are outside the maze
369.                 return false;
370.             } else if (y % 1 !== 0 && x % 1 === 0){
371.                 let adjTile = this.nodes.getNode({y: Math.floor(y), x: x});
372.                 if (adjTile.bottom === 0){
373.                     // if the y coordinate .5, and the tile above has no bottom wall
374.                     return true;
375.                 }
376.             } else if (x % 1 !== 0 && y % 1 === 0){
377.                 let adjTile = this.nodes.getNode({ y: y, x: Math.floor(x)});
378.                 if (adjTile.right === 0){
379.                     // if the x coordinate is .5 and the tile to the left has no
   right wall
380.                     return true;
381.                 }
382.             }
383.             return false;
384.         }
385.
386.         /**
387.          * If the tile is an edge, and it doesn't contain a piece, then delete it
   from the used edges object. Otherwise, add it
388.          * to the used edges object
389.          * @param tile - the tile object that is being checked
390.          */
391.         checkUsedEdge(tile){
392.             if (tile.type === 'edge'){
393.                 if (tile.contains === undefined){
394.                     // concatenate y and x values to create a unique key for any
   edge
395.                     delete this._usedEdges[String(tile.y)+String(tile.x)];
396.                 } else {
397.                     this._usedEdges[String(tile.y)+String(tile.x)] = tile;
398.                 }
399.             }
400.         }
401.
402.         /**
403.          * If the tile is in the maze, return the node at that location. If the tile
   is not in the maze, return false
404.          * @param y - The y coordinate of the node you want to get.
405.          * @param x - The x coordinate of the node you want to get.
406.          * @returns A node or edge object.
407.          */
408.         getTile(y, x){
409.             if (this.checkTileInMaze(y, x)) {
410.                 if (x % 1 === 0 && y % 1 === 0){
411.                     // if the x and y values are whole numbers that are within the
   maze's bounds
```

```
                        // then those x and y values can be used to get the correct node from the node list
                        return this.nodes.getNode({y: y, x: x});
                    } else {
                        let edgeUsed = this._usedEdges[String(y)+String(x)];
                        // checking if the tile to get is currently being used to store an item
                        if (edgeUsed !== undefined){
                            return edgeUsed;
                        } else {
                            // if none of the above, then the tile will either be a horizontal or vertical edge
                            if (x % 1 !== 0) {
                                return new HorizontalEdge(y, x);
                            } else if (y % 1 !== 0) {
                                return new VerticalEdge(y, x);
                            }
                        }
                    }
                }
                return false;
            }

            /**
             * The output function will print out the adjacency list to the console
             */
            output(){
                console.log(this.nodes);
            }
        }


        /* It's a class that represents an entity in the game */
        class Entity {
            /**
             * The constructor function is used to create a new object with the properties of the class
             * @param x - The x coordinate of the entity.
             * @param y - The y coordinate of the entity
             */
            constructor(x, y) {
                this.x = x;
                this.y = y;
                this._speed = 0;
                this._prevTile = {y: 0, x:0};
                this._currentTile = {y: 0, x:0};
                this._tileOrigin = {y: 0, x:0};
                this.moveDirections = [];
                this._attackCooldown = 0;
                this._attackDamage = 0;
            }

            /**
             * If the decimal portion of the tile coordinate is greater than 0.5, round down to the nearest half-tile. Otherwise,
             * round down to the nearest tile
             * @param tileCoord - The tile coordinate to round.
             * @returns The rounded tile coordinate.
             */
            roundTileCoord(tileCoord) {
                if (tileCoord - Math.floor(tileCoord) > 0.5) {
                    tileCoord = Math.floor(tileCoord) + 0.5;
```

```
469.                 } else {
470.                     tileCoord = Math.floor(tileCoord);
471.                 }
472.                 return tileCoord;
473.             }
474.
475.             /**
476.              * The function determines the current tile the entity is in by rounding the entity's x and y coordinates to the
477.              * nearest tile
478.              */
479.             determineCurrentTile(){
480.                 // store previous node to know where its walls where
481.                 this._prevTile.x = this._currentTile.x;
482.                 this._prevTile.y = this._currentTile.y
483.
484.                 // work out which tile in the spanning tree the entity is in
485.                 let x = this.roundTileCoord((this.x / mazeScale) + 1);
486.                 let y = this.roundTileCoord((this.y / mazeScale) + 1);
487.
488.                 this._currentTile = game.maze.getTile(y, x);
489.                 this.determineTileOrigin();
490.             }
491.
492.             /**
493.              * This function determines the origin of the tile that the entity is currently on
494.              */
495.             determineTileOrigin(){
496.                 // get the coordinates of the tile and data from spanning tree
497.                 this._tileOrigin.x = (this._currentTile.x - 1) * mazeScale;
498.                 this._tileOrigin.y = (this._currentTile.y - 1) * mazeScale;
499.             }
500.
501.             /**
502.              * If the entity is within the bounds of a tile, check if the entity is colliding with a wall. If so, move the entity
503.              * back to their original position
504.              * @param originalX - the original x position of the entity before the move
505.              * @param originalY - the original y position of the entity before the move
506.              */
507.             checkCollision(originalX, originalY) {
508.                 // maze wall collisions
509.                 // top, bottom, left, right
510.
511.                 this.determineCurrentTile();
512.
513.                 // left
514.                 if (this.x < this._tileOrigin.x + 3) {
515.                     if (this._currentTile.left === 1) {
516.                         this.x = originalX;
517.                         // checking the corners on the left side of the tile if there is no left wall
518.                     } else if (this._currentTile.left === 0 && (this.y > this._tileOrigin.y + 46 || this.y < this._tileOrigin.y + 5)) {
519.                         this.y = originalY;
520.                     }
521.                 }
522.                 // right
523.                 if (this.x > this._tileOrigin.x + 45) {
524.                     if (this._currentTile.right === 1) {
525.                         this.x = originalX;
```

```
526.                     // checking the corners on the right side of the tile if there
   is no right wall
527.                 } else if (this._currentTile.right === 0 && (this.y >
   this._tileOrigin.y + 46 || this.y < this._tileOrigin.y + 5)) {
528.                     this.y = originalY;
529.                 }
530.             }
531.             // top
532.             if (this.y < this._tileOrigin.y + 5) {
533.                 if (this._currentTile.top === 1) {
534.                     this.y = originalY;
535.                     // checking the corners on the top side of the tile if there is
   no top wall
536.                 } else if (this._currentTile.top === 0 && (this.x <
   this._tileOrigin.x + 3|| this.x > this._tileOrigin.x + 45)) {
537.                     this.x = originalX;
538.                 }
539.             }
540.             // bottom
541.             if (this.y > this._tileOrigin.y + 46) {
542.                 if (this._currentTile.bottom === 1) {
543.                     this.y = originalY;
544.                     // checking the corners on the bottom side of the tile if there
   is no bottom wall
545.                 } else if (this._currentTile.bottom === 0 && (this.x <
   this._tileOrigin.x + 3|| this.x > this._tileOrigin.x + 45)) {
546.                     this.x = originalX;
547.                 }
548.             }
549.         }
550.
551.         /**
552.          * If the entity is moving, check if the new position is blocked, and if
   not, move the entity to the new position
553.          */
554.         move() {
555.             if (this.moveDirections.length > 0){
556.
557.
558.                 // storing position from previous frame in case new position is
   blocked
559.                 let originalX = this.x;
560.                 let originalY = this.y;
561.
562.
563.                 this.determineCurrentTile();
564.
565.                 for (let i = 0; i < this.moveDirections.length; i++) {
566.                     switch (this.moveDirections[i]) {
567.
568.                         case directions.right:
569.                             this.x += this._speed;
570.                             break;
571.
572.                         case directions.left:
573.                             this.x -= this._speed;
574.                             break;
575.
576.                         case directions.down:
577.                             this.y += this._speed;
578.                             break;
579.
```

```
580.                         case directions.up:
581.                             this.y -= this._speed;
582.                             break;
583.                     }
584.                 }
585.
586.                 this.checkCollision(originalX, originalY);
587.                 // animations change based on HTML attributes
588.                 this.self.setAttribute("facing", this.moveDirections[0]);
589.                 this.self.setAttribute("walking", "true");
590.
591.             } else {
592.                 this.self.setAttribute("walking", "false");
593.             }
594.         }
595.
596.         /**
597.          * This function returns the current tile
598.          * @returns The current tile that the entity is on.
599.          */
600.         getCurrentTile(){
601.             return this._currentTile;
602.         }
603.
604.         /**
605.          * If the entity is not currently attacking, set the attacking attribute to
    true, set the facing attribute to the
606.          * direction of the target, damage the target, and set the attacking
    attribute to false after the attack cooldown.
607.          * @param target - The target to attack.
608.          */
609.         attack(target){
610.             // can only run if the entity is not already attacking
611.             // i.e. they are not in an attack cooldown
612.             if (this.self.getAttribute("attacking") !== "true"){
613.
614.                 this.self.setAttribute("attacking", "true");
615.
616.                 // checks which direction the target is from the entity,to play the
    correct animation
617.                 if (target.x <= this.x){
618.                     this.self.setAttribute("facing", "left");
619.                 } else if (target.x > this.x) {
620.                     this.self.setAttribute("facing", "right");
621.                 }
622.
623.                 //dealing damage to the target
624.                 target.damage(this._attackDamage)
625.
626.                 // sets the attack cooldown - the entity won't be able to attack
    again until this is over
627.                 window.setTimeout(function (self){
628.                     self.setAttribute("attacking", "false");
629.                 }, this._attackCooldown, this.self);
630.             }
631.         }
632.     }
633.
634.
635.     /* It's a class that represents an item in the game */
636.     class ItemEntity extends Entity{
637.         /**
```

```
638.          * The constructor function sets the object's x and y coordinates,
   determines
639.          * the current tile, and creates a div element for the object
640.          */
641.         constructor() {
642.             super(20, 28);
643.             this.determineCurrentTile();
644.             this.self = document.createElement("div");
645.             this.self.className = "item";
646.         }
647.
648.         /**
649.          * It takes the entity's x and y coordinates, and determines which tile in
   the spanning tree the entity is in
650.          * @param y - the y coordinate of the entity
651.          * @param x - the x coordinate of the entity
652.          */
653.         determineCurrentTile(y, x){
654.             // work out which tile in the spanning tree the entity is in
655.             let tileX = this.roundTileCoord((x / mazeScale) + 1);
656.             let tileY = this.roundTileCoord((y / mazeScale) + 1);
657.
658.             this.determineTileOrigin(tileY, tileX);
659.         }
660.
661.         /**
662.          * "Given a tile's coordinates, determine the coordinates of the tile's
   origin."
663.          *
664.          * @param tileY - the y coordinate of the tile
665.          * @param tileX - the x coordinate of the tile
666.          */
667.         determineTileOrigin(tileY, tileX){
668.             // get the coordinates of the tile and data from spanning tree
669.             this._tileOrigin.x = (tileX - 1) * mazeScale;
670.             this._tileOrigin.y = (tileY - 1) * mazeScale;
671.         }
672.
673.         /**
674.          * > The function places the object on the map
675.          */
676.         place(){
677.             this.y = this._tileOrigin.y+32;
678.             this.x = this._tileOrigin.x+24;
679.
680.             // adding item entity's HTML element into the document
681.             game.map.appendChild(this.self);
682.             this.updatePosition();
683.         }
684.
685.         /**
686.          * The function determines the tile's origin and then places the tile
687.          * @param tileY - The y coordinate of the tile you want to spawn on.
688.          * @param tileX - The x coordinate of the tile you want to spawn on.
689.          */
690.         spawn(tileY, tileX){
691.             this.determineTileOrigin(tileY, tileX);
692.             this.place();
693.         }
694.
695.         /**
```

```
696.           * The function move() takes in two parameters, y and x, and then determines
   the current tile,
697.           * and then calls the function place().
698.           * @param y - the y coordinate of the tile you want to move to
699.           * @param x - The x coordinate of the tile you want to move to.
700.           */
701.          move(y, x){
702.              this.determineCurrentTile(y, x);
703.              this.place();
704.          }
705.
706.          /**
707.           * This function updates the position of the item on the screen.
708.           */
709.          updatePosition(){
710.              this.self.style.transform = `translate3d( ${this.x * pixelSize}px,
   ${this.y * pixelSize}px, 0 )`;
711.          }
712.
713.
714.          /**
715.           * Remove the element from the DOM.
716.           */
717.          remove(){
718.              this.self.outerHTML = "";
719.          }
720.      }
721.
722.
723.      /* It's a class that represents an enemy in the game */
724.      class Enemy extends Entity {
725.          /**
726.           * The constructor function is used to create a new enemy object
727.           * @param id - The id of the enemy.
728.           */
729.          constructor(id) {
730.              super(27, 16);
731.              this.id = id;
732.              this._speed = 7/8;
733.              this._range = 0;
734.              this._maxHealth = 10;
735.              this._health = this._maxHealth;
736.              this._attackCooldown = 1000;
737.              this._attackDamage = 1;
738.              this._path = [];
739.              this._target = {y: -1, x: -1};
740.              this._targetTile = {};
741.              this.self = document.createElement("div");
742.              this.self.className = "enemy";
743.              this.self.id = `enemy-${this.id}`;
744.          }
745.
746.          /**
747.           * The spawn function is called when the enemy is created, and it sets the
   enemy's position on the map, and creates the
748.           * enemy's health bar
749.           * @param tileY - The y coordinate of the tile you want the enemy to spawn
   on.
750.           * @param tileX - The x coordinate of the tile you want the enemy to spawn
   on.
751.           */
752.          spawn(tileY, tileX){
```

```
753.                this.self.setAttribute("attacking", "false")
754.
755.                this.x = (tileX -1) * mazeScale + 20;
756.                this.y = (tileY -1) * mazeScale + 40;
757.                this._currentTile = {y: tileY, x: tileX};
758.
759.                // adding the enemy's HTML element to the document
760.                game.map.appendChild(this.self)
761.
762.                let spriteSheet = document.createElement("div");
763.                spriteSheet.className = "enemy-spritesheet";
764.                this.self.appendChild(spriteSheet);
765.
766.                this.self.style.transform = `translate3d( ${this.x * pixelSize}px,
    ${this.y * pixelSize}px, 0 )`;
767.                this.healhBar = new HealthBar(this.self.id, this._maxHealth);
768.            }
769.
770.            /**
771.             * The function takes in a number, subtracts that number from the enemy's
    health, and if the enemy's health is less
772.             * than or equal to 0, it removes the enemy from the game
773.             * @param damageTaken - The amount of damage the enemy takes.
774.             */
775.            damage(damageTaken){
776.                this._health -= damageTaken
777.                if (this._health <= 0){
778.                    // doesn't allow the enemy's health to drop below 0
779.                    this._health = 0;
780.
781.                    // removes the enemy's HTML element from the document
782.                    this.self.outerHTML = "";
783.
784.                    game.player.enemiesKilled += 1;
785.
786.                    for (let index = 0; index < game.enemyGroup.objectList.length;
    index++){
787.                        let enemy = game.enemyGroup.objectList[index];
788.                        // removing the enemy from the enemy group
789.                        // locates correct enemy by id, then removes by index
790.                        if (enemy.id === this.id){
791.                            game.enemyGroup.objectList.splice(index, 1);
792.                            break;
793.                        }
794.                    }
795.                }
796.                this.healhBar.update(this._health);
797.            }
798.
799.            /**
800.             * The enemy finds a path to the player by checking the tiles around it and
    moving in the direction of the player
801.             */
802.            pathFind(){
803.                // this._range is the diameter of the enemy's searching area
804.                let min = {y: this._currentTile.y - this._range/2, x:
    this._currentTile.x - this._range/2};
805.                let max = {y: this._currentTile.y + this._range/2, x:
    this._currentTile.x + this._range/2};
806.
807.                // enemy should only find a path if the target is within its range
```

```
808.             if (min.y <= this._targetTile.y && this._targetTile.y <= max.y && min.x
   <= this._targetTile.x && this._targetTile.x <= max.x){
809.
810.                 let nodesInRange = new NodeList();
811.
812.                 // adding all the tiles (nodes and edges) that are within the
   enemy's range to a node list
813.                 for (let row = min.y; row <= max.y; row += 0.5){
814.                     for (let column = min.x; column <= max.x; column += 0.5){
815.                         let node = game.maze.getTile(row, column);
816.                         if (node){
817.                             nodesInRange.push(node);
818.                         }
819.                     }
820.                 }
821.
822.                 // recursive backtracking starts at target for better efficiency
823.                 let checkTile = this._targetTile;
824.                 // x and y coordinate of tile saved separately as they will need to
   be referenced often, and the positions
825.                 // will need to be compared
826.                 let checkPosition = checkTile.position();
827.                 let visitedNodes = new NodeList();
828.
829.                 while(true){
830.                     if (checkPosition.y === this._currentTile.y && checkPosition.x
   === this._currentTile.x){
831.                         // has found player
832.                         break;
833.                     }
834.
835.                     // ensures a node is not checked twice
836.                     nodesInRange.delete(checkPosition);
837.                     let nextPosition = checkPosition;
838.                     let direction = "";
839.
840.                     // if the player is in an adjacent tile, the correct direction
   is known
841.                     if (nextPosition.x === this._currentTile.x && nextPosition.y +
   0.5 === this._currentTile.y){
842.                         nextPosition.y += 0.5;
843.                         direction = "up";
844.
845.                     } else if (nextPosition.x === this._currentTile.x &&
   nextPosition.y - 0.5 === this._currentTile.y){
846.                         nextPosition.y -= 0.5;
847.                         direction = "down";
848.
849.                     } else if (nextPosition.y === this._currentTile.y &&
   nextPosition.x - 0.5 === this._currentTile.x){
850.                         nextPosition.x -= 0.5;
851.                         direction = "right";
852.
853.                     } else if (nextPosition.y === this._currentTile.y &&
   nextPosition.x + 0.5 === this._currentTile.x){
854.                         nextPosition.x += 0.5;
855.                         direction = "left";
856.
857.                     // when the player is not in an adjacent tile, tiles are checked
   using recursive backtracking
858.                     } else if (checkTile.top === 0 && nodesInRange.contains({y:
   nextPosition.y - 0.5, x: nextPosition.x})){
```

```
                            nextPosition.y -= 0.5;
                            direction = "down";

                        } else if (checkTile.bottom === 0 && nodesInRange.contains({y:
    nextPosition.y + 0.5, x: nextPosition.x})){
                            nextPosition.y += 0.5;
                            direction = "up";

                        } else if (checkTile.left === 0 && nodesInRange.contains({y:
    nextPosition.y, x: nextPosition.x - 0.5})){
                            nextPosition.x -= 0.5;
                            direction = "right";

                        } else if (checkTile.right === 0 && nodesInRange.contains({y:
    nextPosition.y, x: nextPosition.x + 0.5})){
                            nextPosition.x += 0.5;
                            direction = "left";

                        } else {
                            // can't find player down this route
                            if (this._path.length !== 0) {
                                this._path.shift();
                                // nodes will be popped from list until either a new
    route is found, or the enemy realises it
                                // cannot find the player
                                checkPosition = visitedNodes.pop();
                            } else {
                                break;
                            }
                        }
                        if (nodesInRange.contains(nextPosition)){
                            // adding direction to the path as the route is followed
                            this._path.unshift(direction);
                            // records the order in which tiles have been visited to
    allow backtacking
                            visitedNodes.push(checkTile);
                            checkPosition = nextPosition;
                            checkTile =
    nodesInRange.dict[nodesInRange.getKeyFromPos(checkPosition)];
                        }
                    }
                }
            }

            /**
             * If the player is on the same tile as the enemy, move towards the player.
    If the player is not on the same tile as
             * the enemy, move towards the next tile in the path
             */
            move(){
                this.determineTileOrigin();
                this._targetTile = game.player.getCurrentTile();

                if (this._targetTile.x === this._currentTile.x && this._targetTile.y ===
    this._currentTile.y) {
                    // set player as target
                    this._target.x = game.player.x;
                    this._target.y = game.player.y;
                    this._path = [];
                }

                if (this._target.x !== -1 && this._target.y !== -1) {
```

```
913.                    // moving towards a target
914.                    this.moveDirections = [];
915.
916.                    // horizontal and vertical movement are calculated separately so
    enemy can move diagonally, like the player can
917.                    // horizontal movement
918.                    if (this.x - 2 > this._target.x) {
919.                        this.moveDirections.push(directions.left);
920.                    } else if (this.x + 2 < this._target.x) {
921.                        this.moveDirections.push(directions.right);
922.                    }
923.
924.                    // vertical movement
925.                    if (this.y - 2 > this._target.y) {
926.                        this.moveDirections.push(directions.up);
927.                    } else if (this.y + 2 < this._target.y) {
928.                        this.moveDirections.push(directions.down);
929.                    }
930.
931.                    if (this.moveDirections.length > 0) {
932.                        super.move();
933.                    } else {
934.                        // if the enemy has reached the target, target is removed so
    either:
935.                        // - will be in same tile as player so will not need to follow a
    path
936.                        //- will have reached the centre of a tile whilst following a
    path, can now continue to follow the path
937.                        this._target.x = this._target.y = -1;
938.                        this._path.shift();
939.                    }
940.                }
941.
942.                if (this._path.length === 0){
943.                    // if the path has been exhausted and enemy is not in the same tile
    as the player, the enemy should find a new path.
944.                    this._target.x = -1;
945.                    this._target.y = -1;
946.                    this.pathFind();
947.                }
948.
949.
950.                if (this._path.length > 0 && this._target.x === -1 && this._target.y ===
    -1){
951.                    // following path to move to next tile
952.                    this.moveDirections = [];
953.                    this.moveDirections.push(this._path[0]);
954.                    super.move();
955.
956.                    if (this._currentTile.x !== this._prevTile.x || this._currentTile.y
    !== this._prevTile.y) {
957.                        // when the enemy crosses from one tile to another, moves to the
    centre of the new tile before continuing along path
958.                        this._target = {y: this._tileOrigin.y + 25, x:
    this._tileOrigin.x + 25};
959.                    }
960.                }
961.                this.self.style.transform = `translate3d( ${this.x * pixelSize}px,
    ${this.y * pixelSize}px, 0 )`;
962.            }
963.
964.            /**
```

```
965.            * If the distance between the player and the enemy is less than 5, then the
    enemy will attack the player
966.            */
967.           attack(){
968.               // calculating distance between enemy and player using pythagoras
969.               let distance = Math.sqrt(Math.abs(this.x - game.player.x) **2 +
    Math.abs(this.y  - game.player.y )**2);
970.               if (distance < 5) {
971.                   super.attack(game.player);
972.               }
973.           }
974.       }
975.
976.
977.       /* It's a class that represents the Player in the game */
978.       class Player extends Entity {
979.           /**
980.            * The constructor function is used to set the character's starting
    position, health, speed, attack damage, and attack
981.            * cooldown
982.            */
983.           constructor() {
984.               super(27, 16);
985.               this._prevTile = {y:0, x:0};
986.               this._maxHealth = game.maze.height * game.maze.width;
987.               this._health = this._maxHealth;
988.               this._currentTile = game.maze.getTile(1, 1);
989.               this._speed = 1;
990.               this._attackCooldown = 200;
991.               this._attackDamage = 5;
992.               this.inventory = new Inventory();
993.               this.self = document.querySelector('.character');
994.               this._healthBar = new HealthBar('character-1', this._maxHealth);
995.               this.enemiesKilled = 0;
996.               this.locksOpened = 0;
997.               }
998.
999.           /**
1000.           * The function takes a command as a parameter, and then executes the
    command
1001.           * @param command - The command to execute.
1002.           */
1003.          executeCommand(command){
1004.              // converts commands into function calls
1005.              switch (command){
1006.                  case 'interact':
1007.                      this.interact();
1008.                      break;
1009.                  case 'drop':
1010.                      this.drop();
1011.                      break;
1012.                  case 'attack':
1013.                      this.attack();
1014.                      break;
1015.              }
1016.          }
1017.
1018.          /**
1019.           * If the player is standing on a tile that contains an object, and that
    object is a lock, then if the player has a key
1020.           * in their inventory, the lock is unlocked, the key is removed from the
    inventory, and the lock is removed from the
```

```
1021.          * tile. If the player is standing on a tile that contains an object, and
   that object is not a lock, and there is
1022.          * space in the player's inventory, then the object is added to the player's
   inventory, and the object is removed
1023.          * from the tile
1024.          */
1025.         interact(){
1026.             if (this._currentTile.contains !== undefined){
1027.                 if (this._currentTile.contains.objectType === 'lock'){
1028.                     // getting references to both the lock and the item in the
   currently active inventory slot
1029.                     let lock = this._currentTile.contains.getSelf();
1030.                     let itemReference =
   this.inventory.getItemFromSlot(game.activeInventorySlot);
1031.
1032.                     if (itemReference.objectType === 'key'){
1033.                         // if the player is holding a key, the lock is unlocked and
   removed from the maze
1034.                         lock.unlock();
1035.                         this._currentTile.contains = undefined;
1036.                         // the key is removed from the player's inventory
1037.                         this.inventory.removeItem(game.activeInventorySlot);
1038.                         // recording the locks opening, and checking if there are no
   locks left to open
1039.                         // win condition can only occur when a lock is unlocked, so
   only needs to be checked in that instance
1040.                         this.locksOpened += 1;
1041.                         game.checkWinCondition();
1042.                     }
1043.
1044.                 } else {
1045.                     // if not interacting with a lock, the item is picked up
1046.                     let itemReference = this._currentTile.contains;
1047.
1048.                     // items can only be picked up if there is space in the
   inventory
1049.                     if (this.inventory.checkSlotsAvailable()){
1050.                         // remove the item from the maze and add it to the player's
   inventory
1051.                         this.inventory.insertItem(itemReference,
   game.activeInventorySlot);
1052.                         this._currentTile.contains.getSelf().take()
1053.                         this._currentTile.contains = undefined;
1054.                         // checking if the current tile is an edge, in which case it
   should be removed
1055.                         // from the list of edges that are currently in use, as it
   no longer contains an item
1056.                         game.maze.checkUsedEdge(this._currentTile);
1057.                     }
1058.                 }
1059.             }
1060.         }
1061.
1062.         /**
1063.          * If the player is holding a sword, find the closest enemy and attack it
1064.          */
1065.         attack(){
1066.             // player can only attack if they are holding a sword
1067.             if (this.inventory.getItemFromSlot(game.activeInventorySlot).objectType
   === 'sword'){
1068.
```

```
                // setting the closest enemy to an arbitrary enemy to begin checking - will only be used if it
                // is in the player's attack range and is actually the closest enemy
                let closestEnemy = game.enemyGroup.objectList[0];
                // the first instance of closestDistance is set to the furthest distance the player should be able
                // to reach enemies in
                let closestDistance = 10;
                // enemyInRange determines whether an attack will occur or not
                let enemyInRange = false;

                for (const enemy of game.enemyGroup.objectList) {
                    let targetTile = enemy.getCurrentTile()
                    // first will check to see if the enemy is even in the same tile as the player
                    if (targetTile.x === this._currentTile.x && targetTile.y === this._currentTile.y) {
                        // finding the distance between the player and the enemy, which is definitely in the same tile as the player,
                        // using pythagoras
                        let distance = Math.sqrt(Math.abs(this.x - enemy.x) **2 + Math.abs(this.y  - enemy.y )**2);

                        if (distance < closestDistance){
                            // replaces the closest distance with new distance and closest enemy with new enemy
                            // if the enemy being checked is the new closest enemy
                            closestDistance = distance;
                            closestEnemy = enemy;
                            enemyInRange = true;
                        }
                    }
                }

                if (enemyInRange) {
                    super.attack(closestEnemy);
                }
            }
        }

        /**
         * If the player is holding an item and the tile they're on doesn't already have an item on it, then drop the item on
         * the tile
         */
        drop(){
            // will only drop an item if there is an item in the current slot and no item in the tile
            if (this.inventory.getItemFromSlot(game.activeInventorySlot) !== undefined && this._currentTile.contains === undefined){
                // removes item from the slot and places it in the tile
                this._currentTile.contains = this.inventory.removeItem(game.activeInventorySlot);
                this._currentTile.contains.getSelf().place(this.y, this.x);
                // if the tile is an edge, it needs to be saved as it now contains an item
                game.maze.checkUsedEdge(this._currentTile);
            }
        }

        /**
```

```
1118.          * The damage function takes in a number and subtracts it from the player's
   health. If the player's health is less than
1119.          * or equal to zero, the player's health is set to zero and the game ends
1120.          * @param damageTaken - The amount of damage the player takes.
1121.          */
1122.         damage(damageTaken){
1123.             this._health -= damageTaken;
1124.             if (this._health <= 0){
1125.                 // prevents health from falling below 0
1126.                 this._health = 0;
1127.                 // game ends if player drops to 0 health
1128.                 game.gameEnd(false);
1129.             }
1130.             this._healthBar.update(this._health);
1131.         }
1132.
1133.         /**
1134.          * The player moves, and the map moves with the player - the player is
   always in the centre of the camera, unless
1135.          * the player reaches the edge of the map
1136.          */
1137.         move(){
1138.             let mapX = this.x;
1139.             let mapY = this.y;
1140.
1141.             this.moveDirections = game.heldDirections;
1142.             super.move();
1143.
1144.             let widthM = game.imgWidth;
1145.             let heightM = game.imgHeight;
1146.             // smooth camera movement - moves the map against the player while the
   player is in the centre of the map
1147.             if (mapX < 112) { mapX = 112; } // left
1148.             if (mapX > widthM - 112) { mapX = widthM - 112; } // right
1149.             if (mapY < 112) { mapY = 112; } // tops
1150.             if (mapY > heightM - 112) { mapY = heightM - 112; } // bottom
1151.             let camera_top = pixelSize * 112;
1152.             let camera_left = pixelSize * 112;
1153.
1154.             // moving the map and player
1155.             game.map.style.transform = `translate3d( ${-mapX * pixelSize +
   camera_left}px, ${-mapY * pixelSize + camera_top}px, 0 )`;
1156.             this.self.style.transform = `translate3d( ${this.x * pixelSize}px,
   ${this.y * pixelSize}px, 0 )`;
1157.         }
1158.     }
1159.
1160.
1161.     /* It's a container for items that can be picked up and used by the player */
1162.     class Inventory {
1163.         /**
1164.          * It creates a new empty array with 5 elements, and assigns it to the
   _contents property of the object
1165.          */
1166.         constructor(){
1167.             this._size = 5;
1168.             this._slotsAvailable = 5;
1169.             this._contents = [undefined, undefined, undefined, undefined,
   undefined];
1170.         }
1171.
1172.         /**
```

```
1173.          * Sets the id of the slot element of the DOM specified to the type
   specified
1174.          *
1175.          * @param slot - The id of the slot you want to change.
1176.          * @param type - The type of item you want to set the slot to.
1177.          */
1178.         _setDocumentInventorySlot(slot, type){
1179.             let slotView = document.getElementById(slot).firstElementChild;
1180.             if (type !== null){
1181.                 slotView.id = type;
1182.             } else {
1183.                 slotView.id = "";
1184.             }
1185.         }
1186.
1187.         /**
1188.          * It returns true if the number of slots available is not equal to zero
1189.          * @returns whether there are slots available or not
1190.          */
1191.         checkSlotsAvailable(){
1192.             return this._slotsAvailable !== 0;
1193.         }
1194.
1195.         /**
1196.          * If the slot is not undefined, return the item in that slot. Otherwise,
   return false
1197.          * @param slot - The slot number of the item you want to get.
1198.          * @returns The item in the slot.
1199.          */
1200.         getItemFromSlot(slot){
1201.             if (this._contents[slot] !== undefined){
1202.                 return this._contents[slot];
1203.             } else {
1204.                 return false;
1205.             }
1206.         }
1207.
1208.         /**
1209.          * If the inventory is not full, the item is added to the first empty slot.
   If the inventory is full, the item is
1210.          * not added
1211.          * @param itemReference - This is the item object that is being added to the
   inventory.
1212.          * @param activeSlot - The slot that the player is currently hovering over.
1213.          */
1214.         insertItem(itemReference, activeSlot) {
1215.             // can only add items to the inventory if there are slots available
1216.             if (this.checkSlotsAvailable()) {
1217.                 let slot = '';
1218.                 for (let index = 0; index < this._size; index++) {
1219.                     // finds the next available inventory slot for the item being
   picked up to be added to
1220.                     if (this._contents[index] === undefined) {
1221.                         this._contents[index] = itemReference;
1222.                         slot = 'slot-' + index.toString();
1223.                         this._slotsAvailable -= 1;
1224.                         break;
1225.                     }
1226.                 }
1227.                 // sets the inventory slot in the document to the corresponding item
   type
1228.                 this._setDocumentInventorySlot(slot, itemReference.objectType);
```

```
1229.             }
1230.         }
1231.
1232.         /**
1233.          * This function removes an item from the inventory and returns a reference
      to the item.
1234.          * @param activeSlot - The slot number of the item to be removed.
1235.          * @returns The reference to the item being removed
1236.          */
1237.         removeItem(activeSlot){
1238.             let itemReference = this._contents[activeSlot];
1239.             this._contents[activeSlot] = undefined;
1240.             this._setDocumentInventorySlot('slot-' + activeSlot.toString(), null);
1241.             this._slotsAvailable += 1;
1242.             return itemReference;
1243.         }
1244.     }
1245.
1246.
1247.     /* It's a class that represents a healthbar in the game */
1248.     class HealthBar{
1249.         /**
1250.          * The constructor function creates a div element with the class name
      "health_bar" and appends it to the parent element
1251.          * @param parentId - The id of the element that the health bar will be
      appended to.
1252.          * @param maxHealth - The maximum health of the entity.
1253.          */
1254.         constructor(parentId, maxHealth) {
1255.             this._maxHealth = maxHealth;
1256.             // finds the element the healthbar will be attacked to in the document
1257.             let parent = document.getElementById(parentId);
1258.             this._self = document.createElement("div");
1259.             this._self.className = "health_bar";
1260.             parent.appendChild(this._self);
1261.             this.update(maxHealth);
1262.         }
1263.
1264.         /**
1265.          * It updates the health bar.
1266.          * @param health - The current health of the entity.
1267.          */
1268.         update(health){
1269.             // the size of a full healthbar should cover half of the entity's width
1270.             this._self.style.width= `${(health/this._maxHealth) * 50}%`;
1271.         }
1272.     }
1273.
1274.
1275.     /* It's a class that handles the game logic */
1276.     class GameController{
1277.         /**
1278.          * It creates a new game object, and initializes all the variables that will
      be used throughout the game.
1279.          */
1280.         constructor() {
1281.             this._gameOver = false;
1282.
1283.             // initialising objects to store the main game objects in
1284.             this.itemGroup = new ObjectGroup('item');
1285.             this.lockGroup = new ObjectGroup('lock');
1286.             this.enemyGroup = new ObjectGroup('enemy');
```

```
1287.
1288.             this.heldDirections = [];
1289.             this.activeInventorySlot = 0;
1290.             // gives the first active inventory slot the indicator image
1291.             document.getElementById(`slot-
   ${this.activeInventorySlot}`).style.backgroundImage =
   "url(/static/img/active_slot.png)";
1292.
1293.             // storing elements that will need to be referenced
1294.             this.map = document.querySelector('.map');
1295.             this.endScreen = document.querySelector('.end-screen');
1296.             this.level = document.querySelector('.level');
1297.
1298.             // counts up in milliseconds from when the page loads
1299.             this.timeElapsed = 0;
1300.             setInterval(function(){
1301.                 game.timeElapsed += 1;
1302.             }, 1000, );
1303.
1304.             // storing elements that will be referenced later
1305.             this._timerStatus = document.querySelector(".time-elapsed");
1306.             this._locksStatus = document.querySelector(".locks-remaining");
1307.             this._enemiesStatus = document.querySelector(".enemies-defeated");
1308.         }
1309.
1310.         /**
1311.          * It updates the game status text on the screen
1312.          */
1313.         _updateGameStatus(){
1314.             // turns the millisecond count into minutes and seconds that will be
   displayed
1315.             let minutes = Math.floor(this.timeElapsed /60);
1316.             let seconds = Math.floor((this.timeElapsed/60 - minutes)*60);
1317.             if (String(seconds).length < 2){
1318.                 seconds = `0${seconds}`;
1319.             }
1320.
1321.             this._timerStatus.textContent = `time elapsed: ${minutes}:${seconds}`;
1322.             this._locksStatus.textContent = `Locks remaining:
   ${this.lockGroup.objectList.length - this.player.locksOpened}`;
1323.             this._enemiesStatus.textContent = `Enemies defeated:
   ${this.player.enemiesKilled}`;
1324.         }
1325.
1326.         /**
1327.          * It changes the background image of the active inventory slot to a
   different image
1328.          * @param slot - The slot number to set as active.
1329.          */
1330.         setActiveInventorySlot(slot){
1331.             if (this.activeInventorySlot !== slot){
1332.
1333.                 let prevSlot = this.activeInventorySlot;
1334.                 this.activeInventorySlot = slot;
1335.
1336.                 let slotView = document.getElementById(`slot-${slot}`);
1337.                 let prevSlotView = document.getElementById(`slot-${prevSlot}`);
1338.
1339.                 // swaps the new slot to the indicator image and removes the image
   from the old slot
1340.                 slotView.style.backgroundImage = "url(/static/img/active_slot.png)";
1341.                 prevSlotView.style.backgroundImage = "none";
```

```
1342.             }
1343.         }
1344.
1345.         /**
1346.          * If any of the locks are locked, return false. Otherwise, end the game and
   return true
1347.          * @returns whether the game has been won or not
1348.          */
1349.         checkWinCondition() {
1350.             for (const lock of this.lockGroup.objectList){
1351.                 if (lock.locked){
1352.                     return false;
1353.                 }
1354.             }
1355.             this.gameEnd(true);
1356.             return true;
1357.         }
1358.
1359.         /**
1360.          * This function loops through the enemySpawnPositions array and creates a
   new enemy object for each position in the
1361.          * array
1362.          */
1363.         spawnEnemies(){
1364.             let id = 0;
1365.             for (const coord of this.enemySpawnPositions){
1366.                 id += 1;
1367.                 // gives each enemy a unique id
1368.                 let enemy = new Enemy(id);
1369.                 this.enemyGroup.push(enemy);
1370.                 enemy.spawn(coord.y,  coord.x);
1371.             }
1372.         }
1373.
1374.         /**
1375.          * The function creates a new player, creates a new sword, adds the sword to
   the item group, and then adds the sword to
1376.          * the player's inventory
1377.          */
1378.         spawnPlayer(){
1379.             this.player = new Player();
1380.
1381.             let sword = new Item('sword', 1);
1382.             let swordReference = this.itemGroup.push(sword);
1383.             this.player.inventory.insertItem(swordReference, 0);
1384.         }
1385.
1386.         /**
1387.          * It takes a binary string and converts it into a 2D array of nodes, each
   of which has a top, bottom, left, and right
1388.          * wall
1389.          */
1390.         defineMaze(){
1391.             this.maze = new Maze(mazeSeed);
1392.
1393.             // populating tree and calculating enemy spawn positions
1394.             let enemyNumber =
   Math.floor(3**(((this.maze.height*this.maze.width)**(1/2))/5));
1395.             if (enemyNumber > maxEnemies && maxEnemies !== -1){
1396.                 enemyNumber = maxEnemies;
1397.             }
1398.             // for index checking to work, enemySpawnSpacing must be even
```

```
1399.             let enemySpawnSpacing =
   Math.floor((this.maze.height*this.maze.width)/enemyNumber);
1400.             enemySpawnSpacing -= enemySpawnSpacing % 2
1401.
1402.             let index = 0;
1403.             let deadEndCodes = ['0111', '1011', '1101', '1110'];
1404.             this.deadEndPositions = [];
1405.
1406.             this.enemySpawnPositions = [];
1407.             let enemyCount = 0.5;
1408.
1409.             for (let row = 1; row <= this.maze.height; row++) {
1410.                 for (let column = 1; column <= this.maze.width; column++) {
1411.
1412.                     let bin = this.maze.binaryString.slice(0,4);
1413.
1414.                     // checking if the node is a dead end and/or a node to be used
   to an enemy spawn
1415.                     if (deadEndCodes.includes(bin)){
1416.                         this.deadEndPositions.push({y:row, x:column});
1417.                         console.log()
1418.                     }
1419.
1420.                     // maths behind which indexes are used to spawn enemies is
   explained in design
1421.                     if (index === enemySpawnSpacing*enemyCount && enemyNumber
   !==  0){
1422.                         this.enemySpawnPositions.push({y:row, x:column});
1423.                         enemyCount += 1;
1424.                     }
1425.
1426.                     // populates maze's node list with the nodes defined in the
   binary string
1427.                     let walls = {top: parseInt(bin[0]), bottom: parseInt(bin[1]),
   left: parseInt(bin[2]), right: parseInt(bin[3])};
1428.                     let node = new Node(row, column, walls);
1429.
1430.                     this.maze.nodes.push(node);
1431.                     this.maze.binaryString = this.maze.binaryString.slice(4);
1432.                     index += 1;
1433.                 }
1434.             }
1435.             this.maze.output();
1436.
1437.             this.determineLockAndKeyPositions();
1438.         }
1439.
1440.         /**
1441.          * It spawns locks and keys in the maze, and stores the references to the
   locks and keys in the maze's node list
1442.          */
1443.         determineLockAndKeyPositions(){
1444.             // spawning locks and keys
1445.             let noDeadEnds = this.deadEndPositions.length;
1446.             if (noDeadEnds > maxLocks && maxLocks !== -1){
1447.                 noDeadEnds = maxLocks*2;
1448.             }
1449.
1450.             let even = 0;
1451.             if ((noDeadEnds)%2 === 0){
1452.                 even = 1;
1453.             }
```

```
1454.
1455.             // iterating from the end of the list, ignoring the first item if there
   is an odd number of items
1456.             // number of dead ends used must be even to ensure there is the same
   number of keys as locks
1457.             for (let index = noDeadEnds-1; index >= noDeadEnds%2; index--) {
1458.
1459.                 let nodePosition = this.deadEndPositions[index];
1460.
1461.                 // dead ends alternate between containing keys or locks
1462.                 if (index % 2 === even) {
1463.
1464.                     // creating key object and adding it to its group
1465.                     let key = new Item('key', index);
1466.                     let keyReference = this.itemGroup.push(key);
1467.
1468.                     // places the key in the correct place in the maze
1469.                     key.entity.spawn(nodePosition.y, nodePosition.x);
1470.                     this.maze.nodes.getNode(nodePosition).contains = keyReference;
1471.
1472.                 } else if (index % 2 !== even) {
1473.
1474.                     // creating lock object and adding it to its group
1475.                     let lock = new Lock(index);
1476.                     let lockReference = this.lockGroup.push(lock);
1477.
1478.                     // places the lock in the correct place in the maze
1479.                     lock.entity.spawn(nodePosition.y, nodePosition.x);
1480.                     this.maze.nodes.getNode(nodePosition).contains = lockReference;
1481.                 }
1482.             }
1483.         }
1484.
1485.         /**
1486.          * It sets the CSS properties of the root element to the width and height of
   the maze image
1487.          */
1488.         defineMazeProperties(){
1489.             // setting css properties to correct values
1490.             this.root = document.querySelector(':root');
1491.             this.root.style.setProperty('--map-width', this.maze.width);
1492.             this.root.style.setProperty('--map-height', this.maze.height);
1493.
1494.             this.imgWidth = (this.maze.width * mazeScale) - 64;
1495.             this.imgHeight = (this.maze.height * mazeScale) - 50;
1496.         }
1497.
1498.         /**
1499.          * It displays a screen with a message and a button to restart or continue
   the game, the locations are different based
1500.          * on whether the player lost or won, and whether they have finished the
   game or not
1501.          * @param hasWon - boolean
1502.          */
1503.         gameEnd(hasWon){
1504.             let continueLocation = "";
1505.             if ((hasWon === true && game.maze.height*game.maze.width >= 625) ||
   gameComplete === 1){
1506.                 // continue button should redirect to gamecomplete page after maze
   of size 25x25 (level 10) has been completed
1507.                 continueLocation = "/gamecomplete";
1508.             }
```

```
1509.
1510.             // replaces the maze and game with the game's end screen
1511.             this._gameOver = true;
1512.             this.level.id = "hidden";
1513.             this.endScreen.id = "shown";
1514.
1515.             // calculates the score based on time elasped, the maze's area, the
   number of enemies defeated, and the number of locks opened
1516.             let score = Math.floor((this.maze.height + this.maze.width)/2 * (((2 **
   -((this.timeElapsed/200) - 10)) + 50) + (this.player.enemiesKilled * 10) +
   (this.player.locksOpened * 15)));
1517.
1518.             // getting correct references to HTML elements
1519.             let popOut = this.endScreen.firstElementChild;
1520.             let message = popOut.firstElementChild;
1521.
1522.             let scoreDisplay = document.querySelector(".score");
1523.             scoreDisplay.textContent = `Score: ${score}`;
1524.
1525.             let restartButton = document.createElement('button');
1526.             popOut.appendChild(restartButton);
1527.
1528.             if (hasWon === true){
1529.
1530.                 // calculates whether to add 2 or 3 to the size of the maze to go to
   the next level
1531.                 let sizeIncrease = 2;
1532.                 if (this.maze.height % 5 === 0){
1533.                     sizeIncrease = 3;
1534.                 }
1535.                 if (continueLocation === ""){
1536.                     // only sets the continue location to the next largest level if
   the player has not completed the game
1537.                     continueLocation = `/play?height=${this.maze.height +
   sizeIncrease}&width=${this.maze.width + sizeIncrease}`;
1538.                 }
1539.
1540.                 // gives the user the option to try the maze they just completed
   again
1541.                 restartButton.textContent = "Try again?";
1542.                 restartButton.onclick = function(){
1543.                     window.location.href = window.location.href;
1544.                 }
1545.
1546.                 let continueButton = document.createElement('button');
1547.                 continueButton.textContent = "Continue?";
1548.                 continueButton.onclick = function() {
1549.                     window.location.href = continueLocation;
1550.                 }
1551.
1552.                 popOut.appendChild(continueButton)
1553.                 message.style.backgroundImage =
   "url(/static/img/levelcomplete.png)";
1554.
1555.             } else {
1556.                 // user must restart from the menu if they failed the level
1557.                 restartButton.textContent = "Restart?";
1558.                 restartButton.onclick = function(){
1559.                     window.location.href = "/";
1560.                 }
1561.                 message.style.backgroundImage = "url(/static/img/gameover.png)";
1562.             }
```

```
1563.         }
1564.
1565.
1566.         /**
1567.          * The gameLoop function is called every frame and it updates the items, and
   locks and allows the player
1568.          * and enemies to move
1569.          */
1570.         gameLoop() {
1571.
1572.             // need to get pixel size every frame as it varies depending on how
   large the browser window is
1573.             pixelSize =
   parseInt(getComputedStyle(document.documentElement).getPropertyValue('--pixel-size'));
1574.
1575.             this.player.move();
1576.
1577.             for (const enemy of this.enemyGroup.objectList){
1578.                 enemy.move();
1579.                 enemy.attack();
1580.             }
1581.
1582.             // items and locks need to be updated every frame in case the pixel size
   has changed
1583.             for (const item of this.itemGroup.objectList){
1584.                 item.update();
1585.             }
1586.
1587.             for (const lock of this.lockGroup.objectList){
1588.                 lock.update();
1589.             }
1590.             // updates time every frame
1591.             this._updateGameStatus();
1592.         }
1593.
1594.         /**
1595.          * The step function calls the gameLoop function, which updates the game
   state, and then calls itself again
1596.          */
1597.         step() {
1598.             if (this._gameOver === true){
1599.                 return;
1600.             }
1601.             // calls the game loop function every frame
1602.             this.gameLoop();
1603.             window.requestAnimationFrame(function () {
1604.                 game.step();
1605.             })
1606.         }
1607.     }
1608.
1609.     game = new GameController();
1610.     game.defineMaze();
1611.     game.defineMazeProperties();
1612.     game.spawnPlayer();
1613.     game.spawnEnemies();
1614.     game.step();
1615.
1616.
1617.
1618.     /* Listening for key presses and then adding the key to the corresponding array.
   */
```

```
    document.addEventListener('keydown', function (e) {

        let direction = directionKeys[e.key];
        let inventorySlot = inventoryKeys[e.key];
        let command = commands[e.key];
        // adds last key pressed to the start of the heldDirections array
        if (direction && game.heldDirections.indexOf(direction) === -1) {
            game.heldDirections.unshift(direction);

        } else if (inventorySlot && game.activeInventorySlot !== inventorySlot-1) {
            game.setActiveInventorySlot(inventorySlot-1);

        } else if (command){
            game.player.executeCommand(command);
        }
    })

    /* Removing the direction from the heldDirections array when the key is released. */
    document.addEventListener('keyup', function (e) {

        let direction = directionKeys[e.key];
        let index = game.heldDirections.indexOf(direction);
        // removes key from helpDirections when it stops being pressed
        if (index > -1) {
            game.heldDirections.splice(index, 1);
        }
    })
```

Jinja templates

public_template.html

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Import the Bootstrap stylesheet -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap.min.css') }}">
    <!-- Import custom stylesheet -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

    <title>{% block title %}{% endblock %}</title>
</head>

<body>

    <main>
        {% block main %}{% endblock %}
    </main>

    <!-- Import Bootstrap bundle -->
```

```
    <script src="{{ url_for('static', filename='js/bootstrap.bundle.min.js') }}"></script>

    {% block script %}{% endblock %}
</body>

</html>
```

level_template.html

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Import the Bootstrap stylesheet -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap.min.css') }}">
    <!-- Import custom stylesheet -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/level.css') }}">

    <title>{% block title %}{% endblock %}</title>
</head>

<body>

    <main>
        {% block main %}{% endblock %}
    </main>

    {% block script %}{% endblock %}
    <!-- Import Bootstrap bundle -->
    <script src="{{ url_for('static', filename='js/bootstrap.bundle.min.js') }}"></script>
    <!-- Import custom JavaScript -->
    <script src="{{ url_for('static', filename='js/app.js') }}"></script>
</body>

</html>
```

index.html

```
{% extends "public/templates/public_template.html" %}

{% block title %}Lost Cause{% endblock %}

{% block main %}

<div class="title">
    <img src="{{ url_for('static', filename='img/lostcausetitle.png') }}" alt="">
</div>
<div class="container">
    {# button takes user to the first level - 3x3 maze #}
    <button onclick="window.location.href='/play?height=3&width=3'">Play?</button>
```

```
</div>


{% endblock %}

{% block script %}
<script>

</script>
{% endblock %}
```

play.html

```
{% extends "public/templates/level_template.html" %}

{% block title %}Lost Cause{% endblock %}

{% block main %}

<div class='level' id="shown">
    <div class='inventory'>
        {# contains elements for each slot that can contain an item #}
        <div class="slot" id="slot-0">
            {# separate slot and slot sprite elements as the slot will display the active slot indicator #}
            {# and the slot_sprite will display the image for the item that is in the slot #}
            <div class="slot_sprite"></div>
        </div>
        <div class="slot" id="slot-1">
            <div class="slot_sprite"></div>
        </div>
        <div class="slot" id="slot-2">
            <div class="slot_sprite"></div>
        </div>
        <div class="slot" id="slot-3">
            <div class="slot_sprite"></div>
        </div>
        <div class="slot" id="slot-4">
            <div class="slot_sprite"></div>
        </div>
    </div>
    <div class='camera'>
        <div class='map' style="background-image: url(/mazeimages/{{ mazeImage | safe }})">
            {# player character will always be in the game, will start standing stationary #}
            <div class='character' id="character-1" facing='down' walking="false" attacking="false">
                <div class="character_spritesheet"></div>
            </div>
        </div>
    </div>
    <div class="game-status">
        {# displays game stats and controlls for the user #}
        level: {{ level | safe }}
        <div class="time-elapsed"></div>
        <div class="locks-remaining"></div>
        <div class="enemies-defeated"></div><br>
        <div class="controls">
            Controls:<br>
```

```
            WASD > move<br>
            E > pick up/use<br>
            Q > drop<br>
            R > attack<br>
            nums 1-5 > select inventory slot<br>
        </div>
    </div>
</div>
<div class='end-screen' id="hidden">
{# end screen will be shown and populated with the correct elements and image when the game ends #}
    <div class="pop-out">
        <div class="message"></div>
        <div class="score"></div>
    </div>
</div>

{% endblock %}

{% block script %}
<script>
  // used with external js, allows jinja to interact with javascript
    let mazeSeed = "{{ mazeSeed | safe}}";
    const maxEnemies = {{ maxEnemies | safe }}
    const maxLocks = {{ maxLocks | safe }}
    const gameComplete = {{ gameComplete | safe }}

</script>
{% endblock %}
```

gamecomplete.html

```
{% extends "public/templates/public_template.html" %}

{% block title %}Lost Cause{% endblock %}

{% block main %}
    <div class="title" id="gamecomplete"></div>
    <div class="container">
        <div class="maze-list">Mazes completed:
        <div class="flex-container"></div>
        </div>
        <div class="custom-maze">Custom Maze:
        <form action="/play" method="get">
            {# form defines parameters for what will be accepted for each input #}
            <label for="height">Height:</label>
            <input type="number" id="height" name="height" size="2" min="3" max="25" required><br>
            <label for="width">Width:</label>
            <input type="number" id="width" name="width" size="2" min="3" max="25" required><br>
            <label for="maxEnemies">Maximum enemies: (-1 for no limit)</label>
            <input type="number" id="maxEnemies" name="maxEnemies" size="3" min="-1" max="999" value="-1" required><br>
            <label for="maxLocks">Maximum locks: (-1 for no limit)</label>
            <input type="number" id="maxLocks" name="maxLocks" size="3" min="-1" max="999" value="-1" required><br><br>
            <input type="submit" value="Submit">
        </form>
```

```
        </div>
    </div>

{% endblock %}

{% block script %}
    <script>
    // simple script to turn the list of mazes stored by the server for the current session into a flexbox
        let seedList = {{ mazeList | safe }};
        window.onload = function(){
            let flexContainer = document.querySelector('.flex-container');
            for (const seed of seedList) {

                let box = document.createElement('div')
                box.className = 'flex-child'

                let link = document.createElement('a')
                link.href = `/play/${seed}`
                link.textContent = `${seed}`

                box.appendChild(link)
                flexContainer.appendChild(box)
            }
        }
    </script>
{% endblock %}
```

style.css

```
* {
    cursor: url(../img/cursor.png) , auto;
    caret-color: transparent;
}

body {
    background-color:#5e1605;
    color: #c9a95d;
    font-family: monospace;
    font-weight: bold;
}

.flex-container{
    display: flex;
    flex-wrap: wrap;
}

.flex-child {
    flex: 1;
    width: 100px;
    height: 25px;
    text-align: left;
    white-space: nowrap;
    overflow: hidden;
    outline-style: outset;
    outline-color: #8c3a27;
    padding: 2px;
    margin-right: 20px
```

```
}

.title {
    color: transparent;
    text-align: center;
    padding-top: 10%;
}

#gamecomplete{
    margin-top: 5%;
    margin-left: 15%;
    image-rendering: pixelated;
    background-size: 100%;
    width: 70%;
    height: 10%;
    background-image: url(/static/img/gamecomplete.png);
}

.container {
    width: 100%;
    padding: 10px;
}

.custom-maze{
    padding: 20px;
    width: auto;
    margin-left:40%;
}

.maze-list{
    text-align: center;

}

button {
    margin-left: 48%;
    background-color: #822510;
    border-color: #8c3a27;
    color: #c9a95d;
    font-family: fantasy;
}

input {
    color: #c9a95d;
    border-color: #8c3a27;
    background-color: #822510;
    caret-color: #c9a95d;
    margin-left: auto;
}
```

level.css

```
:root {
    --pixel-size: 1px;
    --grid-cell: calc( var(--pixel-size) * 64);
}

/* defines pixel size variable based on the width of the browser window */
@media( min-width: 750px ){
```

```
    :root{
        --pixel-size: 2px;
    }
}

@media( min-width: 1000px ) {
    :root {
        --pixel-size: 3px;
    }
}

@media( min-width: 1400px ) {
    :root {
        --pixel-size: 4px;
    }
}

body {
    cursor: url(../img/cursor.png) , auto;
    background-color:#5e1605;
    color: #c9a95d;
    font-family: monospace;
    caret-color: transparent;
}

#shown{
    visibility: visible;
    pointer-events: all;
}

#hidden{
    visibility: hidden;
    pointer-events: none;
}

.title {
    color: #c9a95d;
    text-align: center;
    padding-top: 10%;
}

.end-screen {
    margin: auto;
    overflow: hidden;
    width: 100%;
    height: calc(var(--pixel-size) * 250);
    position: absolute;
}

.pop-out{
    margin-top: calc(var(--pixel-size) * 50);
    margin-left: calc(var(--pixel-size) * 80);;
    text-align: center;
    width: calc(var(--pixel-size) * 300);
    height: calc(var(--pixel-size) * 100);
}

.message{
    image-rendering: pixelated;
    background-size: 100%;
    width: calc(var(--pixel-size) * 300);
    height: calc(var(--pixel-size) * 42);
```

```
    text-align: center;
}

.score{
    font-family: fantasy;
    font-size: large;
}

.game-status{
    position: relative;
    margin-left: calc(var(--pixel-size) * 260);
    width: calc(var(--pixel-size) * 60);
    height: calc(var(--pixel-size) * 20);
}

button {
    margin-top: calc(var(--pixel-size) * 10);
    background-color: #822510;
    border-color: #8c3a27;
    color: #c9a95d;
    font-family: fantasy;
    margin-left: 5px;
}

.level{
    margin-top: calc(var(--pixel-size) * 15);
    margin-left: calc(var(--pixel-size) * 80);
    overflow: hidden;
    position: absolute;
}


      .inventory{
          image-rendering: pixelated;
          background-image: url("/static/img/inventory-bg.png");
          background-size: 100%;
          width: calc(var(--pixel-size) * 30);
          height: calc(var(--pixel-size) * 150);
          float: left;
          position: relative;

      }

      .camera {
          float: left;
          width: calc(var(--pixel-size) * 224);
          height: calc(var(--pixel-size) * 224);
          overflow: hidden;
          background: #502102;
          position:relative;
          margin-left: 10px;
          cursor: none;
      }

      .map {
          image-rendering: pixelated;
          background-size: 100%;
          width: calc(var(--pixel-size) * ((var(--map-width) * 128) - 64));
          height: calc(var(--pixel-size) * ((var(--map-height) * 128) - 50));
          position: relative;
      }

```

```
132.      .health_bar{
133.          background-color: red;
134.          height: calc(var(--pixel-size) * 2 * 0.5);
135.          margin: auto
136.      }
137.
138.      .enemy{
139.          width: calc(var(--pixel-size) * 18 * 0.5);
140.          height: calc(var(--pixel-size) * 16 * 0.5);
141.          position: absolute;
142.          overflow: hidden;
143.      }
144.
145.      .enemy-spritesheet{
146.          image-rendering: pixelated;
147.          background-image: url("/static/img/slime-spritesheet.png");
148.          background-size: 100%;
149.          width: calc(var(--pixel-size) * 18 * 4);
150.          height: calc(var(--pixel-size) * 14 * 0.5);
151.          margin-top: calc(var(--pixel-size) * 1);;
152.          position: absolute;
153.      }
154.
155.      /* walking animation plays only if enemy is not attacking */
156.      .enemy[walking="true"][attacking="false"] .enemy-spritesheet{
157.          animation: walkAnimation 0.6s steps(4) infinite;
158.      }
159.
160.      .enemy[attacking="true"] .enemy-spritesheet{
161.          animation: attackAnimation 0.2s steps(4);
162.      }
163.
164.      .character{
165.          /*color: transparent;*/
166.          position: absolute;
167.          overflow: hidden;
168.          width: calc(var(--pixel-size) * 37 * 0.5);
169.          height: calc(var(--pixel-size) * 29 * 0.5);
170.      }
171.
172.      .character_spritesheet{
173.          image-rendering: pixelated;
174.          background: url('/static/img/player-spritesheet.png') no-repeat no-repeat;
175.          background-size: 100%;
176.          height: calc(var(--pixel-size) * 27 * 2);
177.          width: calc(var(--pixel-size) * 37 * 4);
178.          margin-top: calc(var(--pixel-size) * 1);;
179.          position: absolute;
180.      }
181.
182.      /* character sprite sheet has different row for different directions */
183.      /* only has attacking animations for horizontal directions */
184.      .character[facing="down"][attacking="false"] .character_spritesheet {
185.          background-position-y: calc(var(--pixel-size) * -13.5);
186.      }
187.      .character[facing="left"] .character_spritesheet {
188.          background-position-y: calc(var(--pixel-size) * -27);
189.      }
190.      .character[facing="up"][attacking="false"] .character_spritesheet {
191.          background-position-y: calc(var(--pixel-size) * -40.5);
192.      }
193.      .character[walking="true"][attacking="false"] .character_spritesheet {
```

```
194.          animation: walkAnimation 0.6s steps(4) infinite;
195.      }
196.      .character[attacking="true"] .character_spritesheet{
197.          animation: attackAnimation 0.2s steps(4);
198.      }
199.
200.      .item{
201.          background-size: 100%;
202.          height: calc(var(--pixel-size) * 15);
203.          width: calc(var(--pixel-size) * 15);
204.          image-rendering: pixelated;
205.          position: absolute;
206.          overflow: hidden;
207.      }
208.
209.      /* defines images to be displayed for the inventory slots */
210.      #key {
211.          background-image: url('/static/img/key.png');
212.      }
213.
214.      #lock {
215.          background-image: url('/static/img/lock.png')
216.      }
217.
218.      #sword {
219.          background-image: url('/static/img/sword.png');
220.      }
221.
222.      .slot{
223.          image-rendering: pixelated;
224.          background-size: 100%;
225.          position: relative;
226.          width: calc(var(--pixel-size) * 30);
227.          height: calc(var(--pixel-size) * 30);
228.      }
229.
230.      .slot_sprite{
231.          image-rendering: pixelated;
232.          background-size: 100%;
233.          position: absolute;
234.          width: calc(var(--pixel-size) * 20);
235.          height: calc(var(--pixel-size) * 20);
236.          margin-left: calc(var(--pixel-size) * 5);
237.          margin-top: calc(var(--pixel-size) * 5);
238.      }
239.
240.      .slot[item='key'] .slot_sprite{
241.          background-image: url('/static/img/key.png');
242.      }
243.
244.      .slot[item='sword'] .slot_sprite{
245.          background-image: url("/static/img/sword.png");
246.      }
247.
248.      /* walk animations are the first half of the sprite sheet */
249.      @keyframes walkAnimation {
250.          from {
251.              transform: translate3d(0%,0%,0);
252.          }
253.          to {
254.              transform: translate3d(-50%,0%,0);
255.          }
```

```
    }

    /* attack animations are the second half of the spritesheet */
    @keyframes attackAnimation {
        from {
            transform: translate3d(-50%, 0%, 0);
        }
        to{
            transform: translate3d(-100%, 0%, 0);
        }
    }
```

Images

All of the following images are displayed at their actual size

active_slot.png: 

cursor.png: 

gamecomplete.png: 

gameover.png: 

key.png: 

levelcomplete.png: 

lock.png: 

inventory-bg.png: 

lostcausetitle.png: 

player-spritesheet.png: 

slime-spritesheet.png: 

sword.png:

Maze tile images

all-walls.png: 

base.png: 

bottom-dead.png: 

bottom-left-corner.png: 

bottom-right-corner.png: 

bottom-wall.png: 

debug-tile.png: 

left-dead.png: 

left-right-wall.png: 

left-wall.png: 

no-walls.png: 

right-dead.png: 

right-wall.png: 

top-bottom-wall.png: 

top-dead.png: 

top-left-corner.png: 

top-right-corner.png: 

top-wall.png: