Contents

Project Analysis

Introduction

My client is an avid fan of rogue-like games and has plenty of experience playing them. My client has expressed to me in the past that some of the games that he plays of this type need to be better and need improvements. My client would like to see a fun, fair and replayable game when it comes to me making it. My client emphasized that the game must have replayability to allow the players of the game not get bored after a couple of runs of the game.

Overview

The game is going to be a dungeon-crawler, rogue-like game, with several loot items that evolve the player, which will be in the form of increasing statistics for the player but also modifying the player's attacks/weapons. These items will be used to fight against bosses and grunt-type enemies as you progress through rooms until you reach the boss in the final room. The game will be a 2D, look-down, overhead game where the user is controlled in four directions with the buttons WASD to move in these directions respectively. Despite this, the direction in which the character will be facing will be in the direction that the mouse cursor, so you are able to aim at the enemies and click MOUSE1 when wanting to attack.

The game will have 5 – 8 stages/levels to proceed through while collecting items out of loot chests, which will be randomly generated, to make the player more powerful as they move closer to the

boss level at the end of the stages. In each of the rooms, there will be smaller enemies that you will have to kill before you are able to proceed onwards; if you have been hit by one of these enemies, your health will be lowered: Once your health hits zero and you are dead, there are no chances of doing the level again and will have to ultimately restart again with a new map / boss . Despite this, the bosses at the end will have different powers which will also be randomly generated, each playthrough.

Research

I researched these types of games to find out the pros and cons by looking at the game's reddit pages and reading articles on what people were not happy about, and what people really praised. Some of the comments I saw on the Binding of Isaac reddit were people complaining about items that were either too overpowered or not having much use at all; this shows me that people would want challenge in the game but also items that have an effect. So, when making my game, I will find a middle ground that balances these items out. I will do this by looking at various items across different game and see what attributes/abilities they give to the main character.

I also asked my client is he agrees with these statements and he did, he also added that some of the games needed to make more of the items that were slightly better than others, needed a lower chance of spawning during a play through as he was getting the items way too frequent.

List of similar games I researched:

- *The Binding of Isaac*
- *Darkest Dungeon*
- *Risk of Rain 2*
- *Enter the Gungeon*

Games which are similar

This is a screen shot of the game Binding of Isaac. This is the game, in which I've researched, the game which I've taken most inspiration off, because it resembles what my client and I want the game to look like when creating my own version. However, we want the game to look different and not just a copy of Binding of Isaac.

Advantages	Disadvantages
• Good graphics • Good Map Design • Good Textures • The mini-map at the top is good to show where the player has been and needs to go.	• Could use more accessible items within a single level. • The GUI could show what active items that the character has been buffed with to increase their stats.

• The GUI is correctly placed in a good spot, which does not cover the screen, but also, allows the user to not have to look too far to find their stats. • Textures have good complexity which make it feel different each time; this makes the player feel that there's a lot replayablility.	• Using hearts gives the user no info on the health of the character; if the health was out of 100, then you could see how low the user is and take precautions if so.

This is a screen shot of the game Risk of Rain 2. It shows some good features which, are staples in the rogue-like game genre, which could be added to my game to make it better. Despite this, it is an online game too which my game will not feature as well.

Advantages	**Disadvantages**
• The items are displayed at the top so the user can see what power-up items they have. • The amount of time you play, the harder the game gets; this addition makes the game more versatile rather than having the normal modes of Easy, Medium and Hard. • The abilities are complex and offer the user many abilities to uses to proceed through the levels. • Good boss design with a lot of complexity integrated within the boss object itself.	• Despite this, when the items are hovered over, they do not tell you what they do, I think this is crucial for new players and people who don't have the time to look up what the items do. • The currency is used to open up containers, this system is not every good because the rarer items cost more, I feel that basing items on the chance and probability is much better to get more rarer items. • When playing the game, users have said that when the levels later on come along, there's too much happening on the screen and that they dislike the messiness of the screen.

 This is a screen shot of the game Enter the Gungeon, I have loosely taken inspiration because the game takes the rogue-like genre in a different direction, where I want my game to be a classic rogue-like.

Advantages	**Disadvantages**
• The game is complex and has a lot of replayability. • The game offers items on chance and with in-game currency. • Vast range of textures • The map design is well thought out • There are a lot of different type of enemies for the player to experience. • The enemies have a wide range of abilities to attack the player with	• The RNG within the game for the items is very unbalanced, so you are able to get overpowered items more easily. • The game has poor performance and runs poorly sometimes, making the user-experience bad for someone with a lower spec computer.

Deciding on the path finding algorithm

A* Searching Algorithm

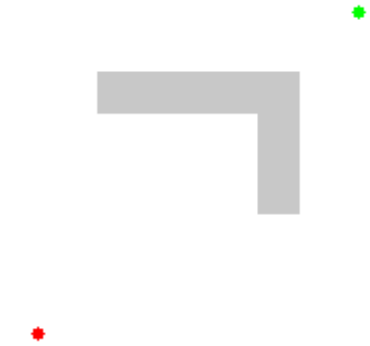
A* search is a graph traversal algorithm, which is mostly used in computer science for its good efficiency. The A* search is formulated on weighted graphs, with a specific starting node and a goal node to head towards. When the A* algorithm wants to extend; it determines what route the path should follow by how much it will cost to make that movement, but also it accounts for the estimate of the cost that is required to extend the path all the way to its goal node. To put it in genral terms, it selects the path the minimizes the path the most.

This would be good to use in a game, because games have a lot of stuff going on at once, therefore, a well-optimised, fast searching algorithm could help the game's performance in allowing the code to run faster, being much smoother for the client to enjoy.

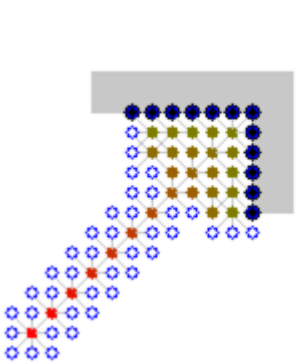
Equation that allows A to function:*

$$f(n) = g(n) + h(n)$$

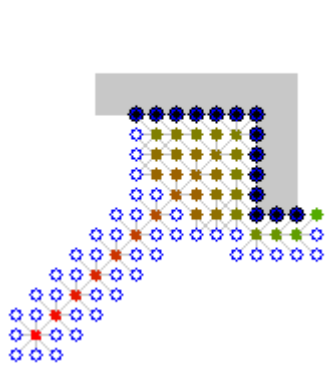
n = Next node in the path
g(n) = The cost from the start node, to the next node n.
h(n) = This is a function where it estimates the cost of the cheapest path from n to the goal node.
f(n) = Both the cost from the start node to the next node added to the cheapest path from n to the goal node. Therefore, being the total cost of the node.



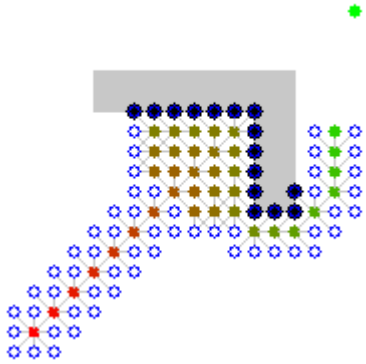
First, the algorithm starts with a set start node and a set goal (end) node. The greyed-out area is a blockade, not allowing the path to go straight to the goal.



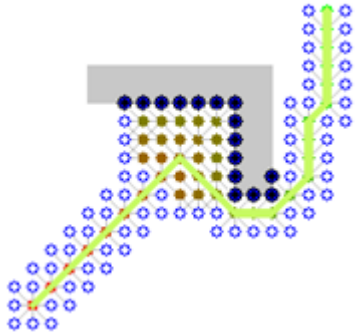
When the node eventually hit the blockade, the algorithm sees that there's no possible way for the node to pass through the blockage; so the algorithm starts to move the nodes downwards to try and overcome this obstacle and find the fastest route to the goal node.



Despite this, the nodes will overcome the obstacle from the shorter side of the blockade to find the quickest path to the goal node without going directly through the blockade.



Once the obstacle has been past, the algorithm then redirects and calculates the fastest route to the goal node.



When the goal node is reached, a path is mapped out from the start to the goal with the quickest route possible.

Pseudocode

let the openList equal empty list of nodes

let the closedList equal empty list of nodes// Add the start node
put the startNode on the openList (leave it's f at zero)// Loop until you find the end
while the openList is not empty // Get the current node
let the currentNode equal the node with the least f value
remove the currentNode from the openList
add the currentNode to the closedList // Found the goal
if currentNode is the goal
if so, backtrack to find the path from the goal node to the initial starting node
let the children of the currentNode equal the adjacent nodes

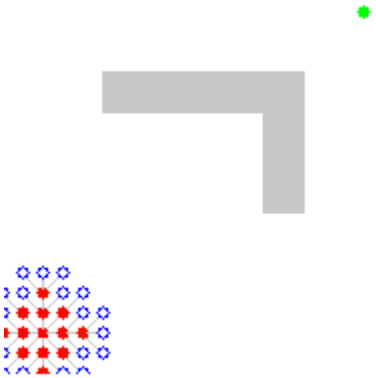
for each child in the children // Child is on the closedList
if child is in the closedList
continue to beginning of for loop // Create the f, g, and h values
child.g = currentNode.g + distance between child and current
child.h = distance from child to end
child.f = child.g + child.h // Child is already in openList
if child.position is in the openList's nodes positions
if the child.g is higher than the openList node's g
continue to beginning of for loop // Add the child to the openList
add the child to the openList

Dijkstra's Searching Algorithm

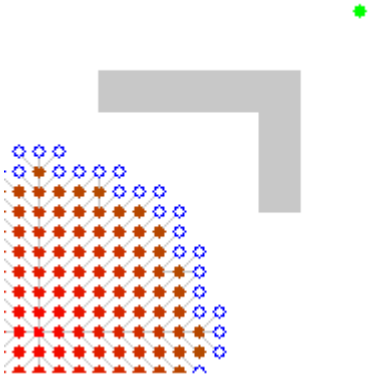
Dijkstra is another graph traversal algorithm; Dijkstra's algorithm is used in many games to help the character find the closest path to traverse when playing. Despite this, Dijkstra's algorithm is slower than A* Search algorithm. But is still considered one of the top searching algorithms when applying to programs.

- Mark all nodes unvisited, then create a set of all the unvisited nodes.
- After this, assign to every node a distance value: set it to zero for the initial node and to infinity for all other nodes.
- For the current node, the algorithm will look at all of its neighbouring nodes and calculate the distance through the current node. Compare the distances of all the nodes that were worked out, and see what route have the shortest value distance.
- When the program has looked at all the neighbouring nodes together, mark the current node as visited; this will make the algorithm skip that current node, as with all the visited nodes.
- If the destination node has been marked visited, then the algorithm has finished, and has found the shortest path between two points.
- If not, select the unvisited node that is marked with the smallest distance, set it as the new current node, and go back to point 3.

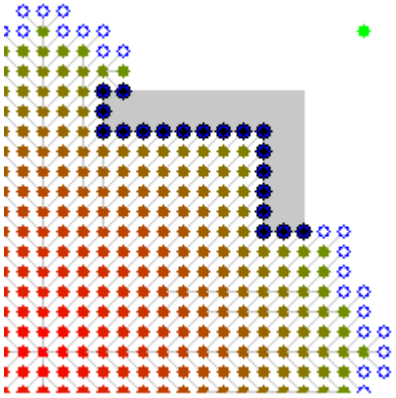
Repeat until the goal node has been marked as visited



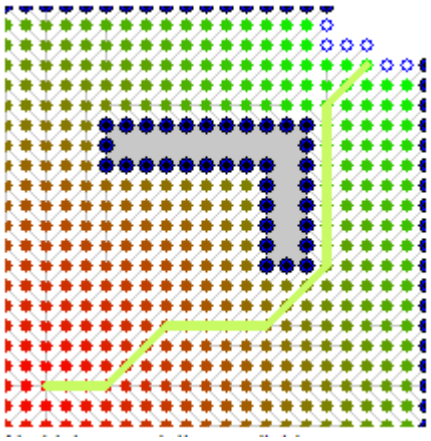
The algorithm starts from and initial node which then looks around itself to all the neighboring unvisited node, so the nodes slowly branch out and get into a much larger shape.



As the algorithm searches, unlike the A* algorithm a lot more nodes are searched across the node array.



When the algorithm hits a blockade, it knows that it needs to go around it. So, the algorithm formulates a path around it to check for the goal node.



Once the goal node is found, a path is formulated between the two point. With the algorithm working correctly, this should be the shortest path between the two points.

Pseudocode

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                 // Initialization
        dist[v] := infinity                     // initial distance from source to vertex v is set to infinite
        previous[v] := undefined                // Previous node in optimal path from source
    dist[source] := 0                           // Distance from source to source
    Q := the set of all nodes in Graph          // all nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:                       // main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:               // where v has not yet been removed from Q.
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]                    // Relax (u,v)
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

Breadth First & Depth First Searching Algorithm

Breadth First Search

With breadth first search, you travel through a tree level at a time when performing tree traversal; so, you travel to the children nodes of the node that you're currently at.

Concern

If this is the searching algorithm that I choose to use in my game when programming, I worry that the program may be slowed down, creating lag for the client when playing; this could be a possibility because the algorithm can take a long time to find the goal node. With programming a game, it's not ideal to hinder the performance and potentially create game-breaking bugs.

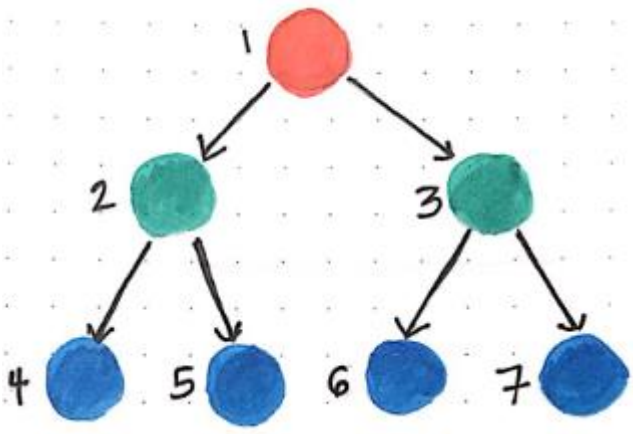


Diagram from: https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

ORANGE is the main starting node.
GREEN are the children nodes of **ORANGE**
BLUE are the children nodes of **GREEN**
So therefore, the order that you traverse the tree with breadth first is:

1. **ORANGE**
2. **GREEN**
3. **BLUE**

Breadth first algorithm is not the fastest when trying to get a fast path finding algorithm, due to it going through all the node to get to a result, which can take a long time.

Depth First Algorithm

With the Depth first searching algorithm you need to search down the left node subtrees, then you search you search the right node subtrees. Which ultimately explores as far down the data structure as possible, then back-tracks to the root node before exploring the right-most children of the root node.

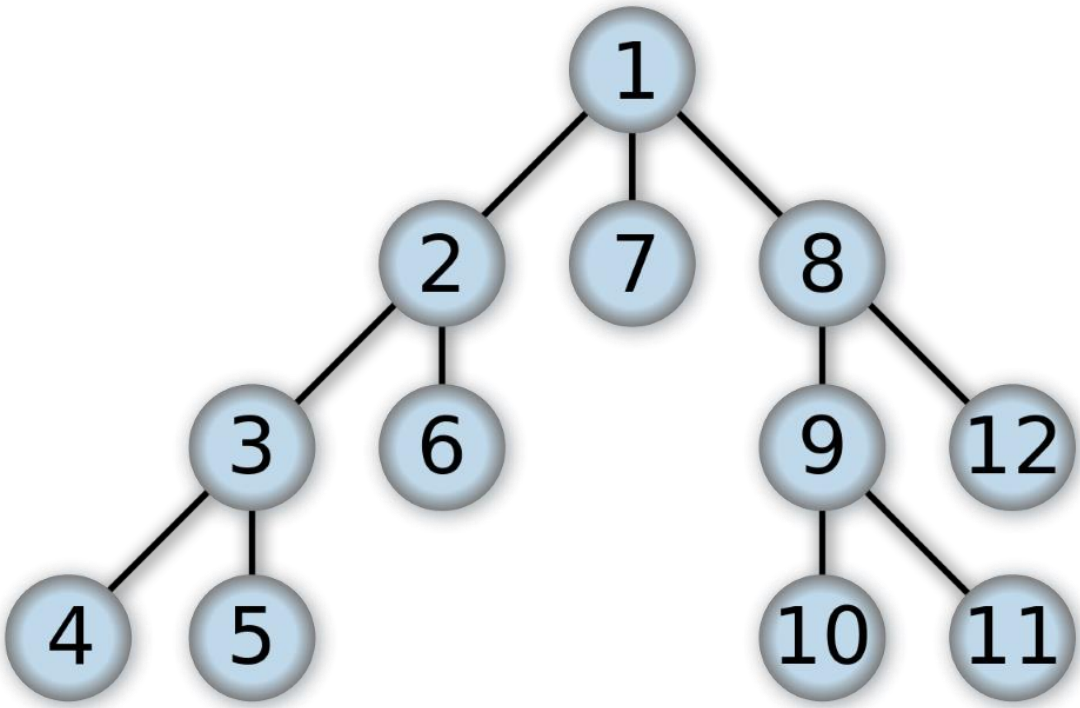


Diagram from: https://en.wikipedia.org/wiki/Depth-first_search

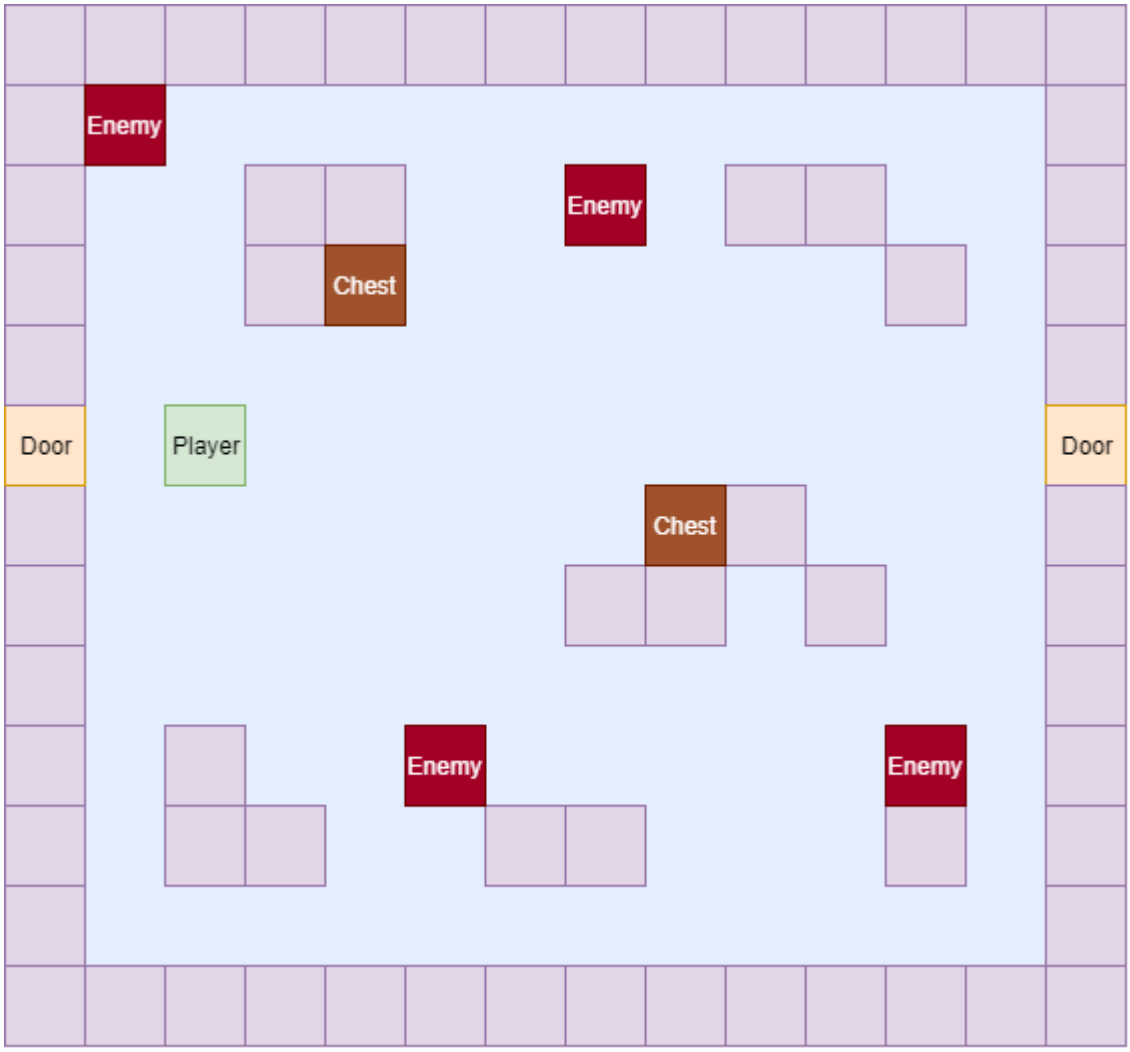
Depth first algorithm is not the fastest searching algorithm either, due to the algorithm having to go through all the nodes and then back track on itself until it finds the correct goal node.

Conclusion for searching algorithm

My client agrees that A* path finding will suit the path finding algorithm for this program. This is due to A* pathfinding being fast and efficient for a game; due to a game having a lot going on, the A* path finding minimizes usage of the game, so the performance is not hindered allowing everything else to function well.

Visual

Rough Visual Representation on what the game will look like:



Input-Process-Storage-Output:

Character Movements:

WASD Keystrokes --- Movement of the main character around the map in the four directions.

When mouse click --- Attack shoots in the direction of the mouse direction when clicked.

Mouse Cursor --- Moved around a 360-degree circle, this faces the main character in that direction.

Data Dictionary

Attribute Name	What is the attribute supposed to do?	Attribute Type	Example Data	
RoomClear	This is essentially a check to see whether the room is clear, and that the player is allowed access to the next room.	Boolean	True	
PlayerHealth	This is to give a visual representation of what the player's health is on.	Integer	100	
PlayerAttackDamage	This is used as a numerical value for the player's damage when attacking enemies.	Integer	10	
PlayerSpeed	This value is used to give the player a numerical value for their velocity when moving around the map.	Integer	5	

RunCompleted	**This is used to indicate that the run has ended and the final boss has been beaten.**	**Boolean**	**True**	
isDead	**This is used to indicate when to end the game when the player has reach a health of zero.**	**Boolean**	**False**	

Data Volumes

MAIN DATA VOLUMES:

- **Attack Damage – This damage will be deducted from an entity's health value.**

- **Health - This will be the amount of health that and entity has; when the value of 0 has been reached, the entity will "die".**

- **Speed – This will be the value of how fast the entity can move around the map.**

- **Character**
  - Base Health (Changeable By Item Buffs)
  - Base Attack Damage (Changeable By Item Buffs)
  - Name
  - Movement Speed (Changeable By Item Buffs)
- **Enemies**
  - Health
  - Attack Damage
- **Boss**
  - Health
  - Attack Damage
  - Sp. Attack Damage – This value will have extended attack damage, due to the move only being able to be used at certain time intervals.
- **Items**
  - Boosts/Buffs given to the character

General basis on how the item system will work and effect enemies when attacking them.

Objects within the program

- **Main Character sprite**
  The Main character will have several abilities which will alter how it performs.
  It will need to send out a projectile when the attack key is push.
  Will also need to store all the character's health, attack damage and speed.
- **Small Enemies**
  Will need to store all the Enemies' health, attack damage and speed.
  AI will be implemented through the enemies to attack the player
- **Boss**
  Will need to store all the Bosses health, attack damage and speed.
  Will need a special attack stored, with all the attributes.
  Each different boss will need to be a separate object in the program.

Client Questionnaire:

[redacted]

As talked about earlier, here are some questions about the project:

Q1. What key aspects of the map layout would you like to see?

Q1b. Would you rather the game to be played as a whole map or have separate levels that the player has to progress through?

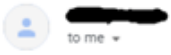
Q2. How would you like the character movement/aiming to be implemented? e.g WASD, arrow keys....

Q3. What improvements would you like to see added from other games you've played?

Q4. What would you like to see from the bosses at the end of the game?

Q5. How would you like the smaller enemies in the game to attack and move around?

Q6. What would you like to see form the overall design of the game?



to me

As talked about earlier, here are some questions about the project:

Q1. What key aspects of the map layout would you like to see?

I would like the map to be different every time when you restart the game, so you are able to have more fun with the game with it not being predictable when you run through it.
I would also like the map to have clear entry and exit points, this is because in some games it is unclear where they initially are and takes some fun out of the game trying to figure it out.

Q1b. Would you rather the game to be played as a whole map or have separate levels that the player has to progress through?

I would rather the game be level based so you can proceed through the game bit by bit without too much happening and being a hectic mess.

Q2. How would you like the character movement/aiming to be implemented? e.g WASD, arrow keys....

I would like the character movement to be moved in all four directions with being able to use the mouse cursor to aim when attacking.

Q3. What improvements would you like to see added from other games you've played?

I would like the items in the game to be fair when picked up, not turning the player into an over-powered character or making little to no difference to the game play when playing.
I would like the game to also have a lot of replayability so the run does not get boring after the first couple of runs.
I would also like the game to have a decent level of difficulty when playing because this, in my case, makes the games I play more fun
I would like the game to have no currency/shop based items because I feel this ruins most of the other games that I have played, I rather the items be added to the game randomly so you never know what you are going to get.

Q4. What would you like to see from the boss at the end of the game?

I would like the boss to have a good level of difficulty, so it's not too easy to beat.
I would also like the boss to have stages to it and abilities so it makes the end fight a lot more fun.
It would also be good to see more than one boss to make the game that bit more unpredictable so you don't know what boss you are going to come up against.

Q5. How would you like the smaller enemies in the game to attack and move around?

I would like the small enemies to be melee characters to allow good strategies and methods when playing each level, so you are essentially avoiding getting hit while fighting them as well.

Q6. What would you like to see form the overall design of the game?

I would like to see a simple, easy to understand game with some cool hidden complexities behind it, in which, is fun to play and has a lot of replayability.

Aims and Objectives:

1. *Constructing the map so that it's different each time.*

a. *Layout a map*

b. *b. Add a wall entity which is placed all around the edge of the map, to act as a barrier for the room.*

c. Having the map be randomly generated, with wall entities in the middle of the room to give the player some cover and diversity to the layout, to provide a different experience for the user each time.

d. Add a texture to the wall entity

This is one of the main objectives due to my client requesting that "the map to be different every time when you restart the game" in the questionnaire email. Despite this, when researching a lot of rogue-type, dungeon crawler type games; I found out that most of these games' maps are procedurally generated with a random number of objects spawned within.

2. *Add the main character sprite.*

a. *Make sure the movement of the character can be moved with the buttons WASD, also make sure that there's smooth movement when walking at the base speed in all directional axis.*

b. Make the character sprite look in the direction which the mouse cursor is facing, allowing the player to move around in 360 degrees.

c. Make the object for the character have base Health, Attack damage and speed statistics

d. Add Main Sprite Texture.

e. Animate the sprite with several images.

This objective is imperative to allow the rest of the game to function properly and allow the other objectives to be fulfilled. This is because the main character can act as an object that is able to interact with other objects within the program.

3. *Add a normal attack for the sprite.*

a. *When the MOUSE1 button is clicked make sure that the character attacks and sends out a projectile.*

b. *Make sure the projectile is shot out the way that the player is facing.*

c. *Projectile must have a texture.*

This objective is added through researching other types of rogue-like games, the general trend is that most of them always have a normal attack which allows the player to defend themselves from enemies.

4. *Make sure Collisions between the characters and map work.*

a. *When a projectile is shot, make sure it is stopped by a wall and doesn't go through it.*

b. *Make sure that the character and other entitles cannot no-clip through the wall and or get out of the map in any way.*

c. *Detection when a projectile hit an enemy, make sure the correct stats are subtracted.*

d. *When a loot box is walked over, a collision should be detected and drop the player an item.*

This objective is more of a necessity in the game, to make sure that the player is unable to glitch in and out of the map or create bugs within the game to make the user's experience not enjoyable.

5. *Procedurally generate items within the maps.*

a. *Make sure to design fair and balanced items.*

b. *Make sure to randomly add items into the map*

c. *When an item is dropped, make sure to correctly buff the player's stats.*

d. *Add textures for different items.*

This is an objective that falls under the same reasoning as objective #1, furthermore my client said that he wanted the items to be fair and balanced, so the user does not easily beat the game or become too difficult for them to complete.

6. *Addition of enemies with pathfinding capabilities.*

a. *Have enemy objects suited with base stats like Health, Attack damage and Speed.*

b. Have an algorithm(s) in place to allow the enemies to path find towards the player to attack it when the collision happens.

This objective follows my clients request of the enemy sprites to be "melee characters", to do this I researched some possible pathfinding algorithms, so the enemies are able to track down the player sprite and path find in the direction towards the player object.

7. *Addition of End Boss Level.*

a. *Implementation of boss.*

b. *Make sure the boss is suited with its own base stats and special attacks.*

c. *Make sure that each boss has a texture,*

d. *Once the boss has been defeated, display game over.*

This objective is mostly based off video games in general, as at the end level there always must be a boss for the player to go against, despite this, while researching; rogue-like games usually have a high demand for bosses, which ultimately makes the game more difficult. Furthermore, I also took my clients suggestion into an account, where he said the boss should be at a decent level of difficultly, so it's no too easy to beat.

Design Section

Overall Design

The program is going to consist of an application which is executed, then a game window will pop up with the menu screen with a play button in the middle. Once the play button has been pressed, the game will start.
Despite this, the user will be placed in an empty room with one door to enter the next room when they're ready to proceed, in the room there will also be instructions and control diagrams in the room to tell the user how to play the game. Once the door has been walked through, and the program has detected the collision, the player will be placed in a procedurally generated room and thrown right into the action with enemies, loot chests and wall entities, with doors to proceed to the next room once all the enemies have been killed.
After proceeding through all 5 rooms the sixth room will be a boss room, in which, the user will have to take the boss on in a head to head fight to complete that game. When fighting the boss, the player should have items that make them more powerful than the starting room; these items will be based on luck with some of the more higher end items having lower odds of acquiring.
Once the boss has been defeated a screen will appear with the time which the player has completed the game in and what items they have.

Window dimensions = 1024x768
Node/Tile size = 64

Coding type

Defensive Programming

Defensive programming is a form of programming that the programmer uses to code for unseen circumstances, these practices include high availability, safety and security.

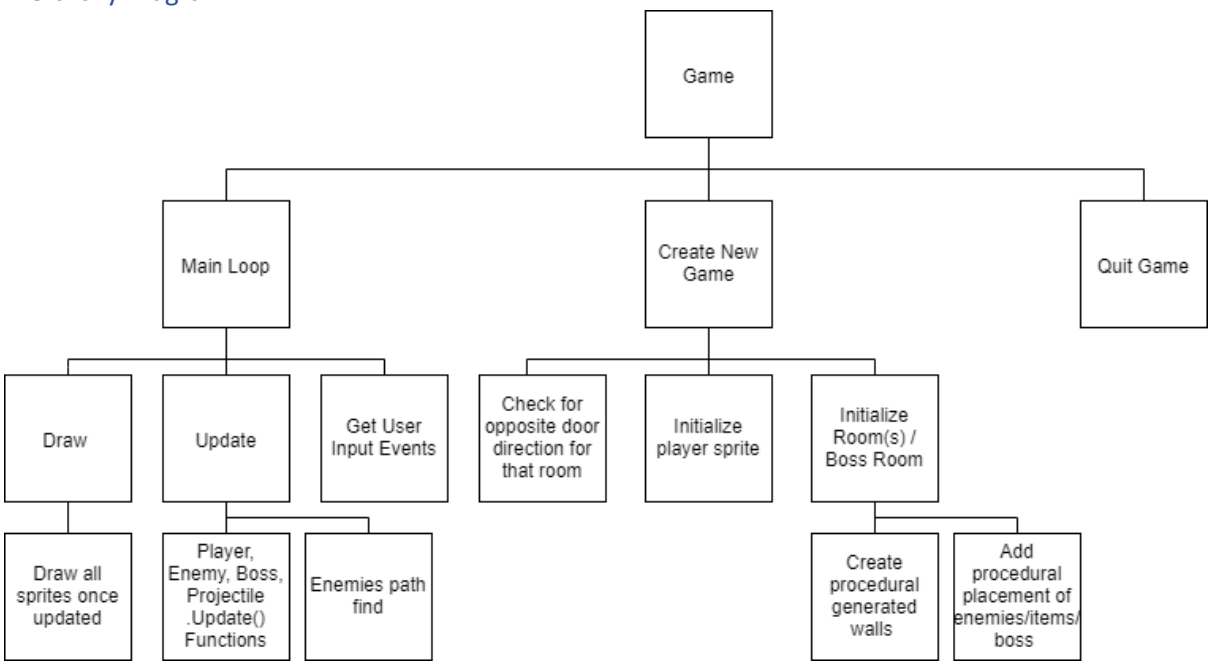
When coding I will use the method of defensive coding to allow the game to have very little bugs, by thinking ahead of the problems that can arise and making sure that the problems do not affect the final product for the client. This style of programming will predominately be the main style when coding.

Offensive Programming

Offensive programming is a form of programming that is the opposite to defensive programming and its practices.

I will also add parts of offensive programming into the code due to the use of
integer coordinates. When using this style this will allow the program to use exception handling to get around the problems that arise when the user's inputs become erroneous.

Hierarchy Diagram



Shows the main stages of the program that will be required to run and function correctly...

Module	***Function***
Main Loop	*The main loop will act as the main part of the game, keeping the program updated with each pass, while on the lookout for any user inputted events to logged; then drawing to the screen so the player can see what's going on*
Create New Game	*When creating a new game, the program will initialize all 5 rooms plus the boss room and initialize the player sprite into the beginning room with coordinates stored for each.*
Quit Game	*When the "x" button is pressed, it will stop the program in its path and exit out the game.*
Check for opposite door direction	*This will be a small function to allow the program to see the opposite side of the room, so rooms do not repeat in the same direction and break the program when going through doors to other rooms.*
Initialize Player sprite	*This will initialize the player into the map with set starting coordinates and defining all attributes.*
Initialize Rooms/Boss Rooms	*Adds enemies, items and walls with stored coordinates and sprite groups for that specific room.*
Draw	*The draw function will be one of the main functions acting under the main*

	loop module, this will be the last to be called when looping through. This is because everything needs to be drawn at the end after everything has updated.
Update	*The update function will also be one of the main functions acting under the main loop module, this function will oversee making sure that the player, enemies and boss are all updating so the game can function and update the positions of the sprites.*
Get user inputted events	*The get events function will also be one of the main functions acting under the main loop module, this function will be checking to see what the user of the program has inputted. E.g. clicks of the mouse, click of the exit cross.*
Path find	*This function will be a big one for the enemies and will be located under the update function. This function will find the quickest path from one point to another to allow the sprites to follow the player around the map*
Create Procedural Generated walls	*This function will work under every room to allow each room to be different.*
Add placement of Enemies/Items/Boss	*This is like the walls where items and random enemies will be placed in the room for the player to overcome*

Data Dictionary

Data dictionary for the player sprite.

Name	**Data Type**	**Description**
Health	Integer	The health will be misused respectively when the sprite is collided with another object.
Start_pos_x	Integer	This will be the starting position for the x coordinate for the player sprite to be initialized into.
Start_pos_y	Integer	This will be the starting position for the y coordinate for the player sprite to be initialized into.
HitBoxRect	Pygame.rect	This will be the hitbox for the player sprite, this will be in charge of detecting the collisions between the player and other objects.

Player_center	Integer	This will be used for different types of positional vectors and detection for the node that the player is in.
Velocityx	Integer	This will allow the player to move across the screen in the x axis.
Velocityy	Integer	This will allow the player to move vertically up and down the screen along the y axis
Image	Image	This will allow the sprite to be seen when moving around the map.

Main Classes

The Program will be written with OOP(Object Oriented Programming), these objects will allow the game's objects to use Inheritance, Abstraction, Polymorphism and Encapsulation when programming.

OOP will allow the entities within the game, E.g. player, enemies, walls, to be stored as their own separate objects with their own attributes and methods. This will be useful to allow each object to interact with each other during collisions. Furthermore, different attributes can be altered within one of these objects to change how that object is stored. E.g Health, when zero the player dies.

I will be using OOP on all the named objects below; this is because these main objects will have many methods and attributes to be stored within to allow each object to interact with each other and allow objects themselves to alter.

PlayerSprite()

This will be the player object; it will have methods and attributes to allow the client to do several things in the game.

Method	**Comments**
Function : get_player_center_coordinates	This function will return the x and y coordinates of the player's position as a tuple.
Function : update	This function will return what room the player is into the main function that the player is based in.
Procedure : movement_keys	This procedure will be called in the update function of the player sprite to allow the player to move by changing its velocity in both x and y directions when a button is pressed that represents all four directions.
Procedure : Rotate	This procedure will allow the player to rotate towards the mouse's direction.
Procedure : Draw	This procedure allows the player to be represented (drawn) onto the screen, once it has updated positions/orientation.

Room()

The room object will initialize many of the other objects within the program and store their data within this object. For example, x and y coordinate, group and the update function for most of

the objects. All the rooms before the boss room will use this but the boss room with use a form of inheritance to allow for different methods and attributes.

Method	**Comments**
Function : Room_Node_check	This function will return what node the player sprite is in, in that current room.
Function : update	This function is going to return a integer value to act as a pointer to the next or previous room. So, the program is able to detect what room it should load next.
Procedure : AddItems	This procedure will randomly add items into each room when initialized with random coordinates.
Procedure : AddEnemies	This procedure will randomly add enemies into each room when initialized with random coordinates.
Procedure : AddWalls	This procedure will procedurally generate different combinations of walls into a room when initialized. It will also add a boundary of walls around the edge of the map.
Procedure : Draw	This will draw all current sprites/items/walls that are stored in the room's sprite groups.

Enemy()

The Enemy object will be opposing the player sprite and try to stop it from getting to the end of the game, the enemies will have their own attributes and methods to give the game complexity when proceeding through it.

Method	**Comments**
Function : GetCenter	This function will return the enemie's x and y coordinates as a tuple.
Function : getNode	This will return the enemies node position, to figure out their current node, this will be used for the enemies path finding.
Function : A Star Pathfinding Search	This function will return the shortest path from two nodes, and will update when the two node positions change.
Function : Breadth First Search	This function will also return the shortest path from two nodes from one static goal node to another dynamic node which the enemy is moving in.
Function : Update	This is where the enemy's movements will be updated and were pathfinding will also take place, to allow an enemy to follow a path.
Procedure : Rotate_towards_player	This procedure will update the enemies' orientation towards the direction of the player.
Procedure : Draw	This will draw the enemy sprite on the screen.

Boss

The boss object will be initialized in the final room of the game and will be the hardest enemy in the game. It will have its own attributes and methods.

Method	**Comments**
Function : Update	This function will update the boss' attack cycle and position to the correct coordinates.

Procedure : Shootprojectiles	This procedure will allow the boss to shoot out projectiles towards the player direction.
Procedure : Attack	This procedure will allow the boss the attack the player in a certain way, once it has been called upon.
Procedure : Draw	This will draw the boss sprite on the screen.

Game()

This will be the main object for the whole program with the main methods/attributes that will be used overall.

Method	**Comments**
Function : returnGameScreen	This function will return the screen background of the game.
Procedure : CreateNewGame	This procedure will initialize the player into the first room. Despite this, it will also initialize each room ready for the player to enter once all the enemies have been killed.
Procedure : Draw_Image	This procedure will be the main draw procedure for each room and every sprite in the game.
Procedure : getInputs	This procedure will oversee detecting all the inputs from the user and categorise each input into a correct method.
Procedure : update	This procedure will be the main update procedure for each room and every sprite in the game.

Main Character Design

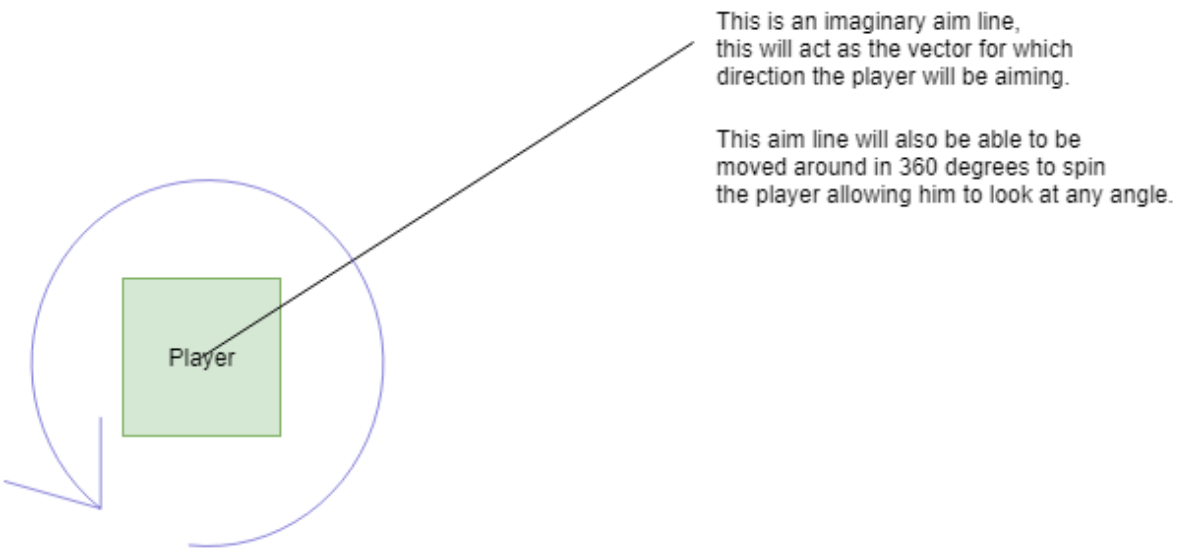
The main character's name is Derek The Slime; He will need to proceed through 6 levels in order to complete his journey, the first five levels will have loot chests, that contain items to power Derek up and give him boosts in his stats, and enemies scattered around trying to stop Derek from getting to final stage at the end. Once Derek has worked his way through all the previous levels, he will be met with his final challenge, which will be a boss that he will need to defeat.

HP – 100
Attack Damage – 15
Speed - 5

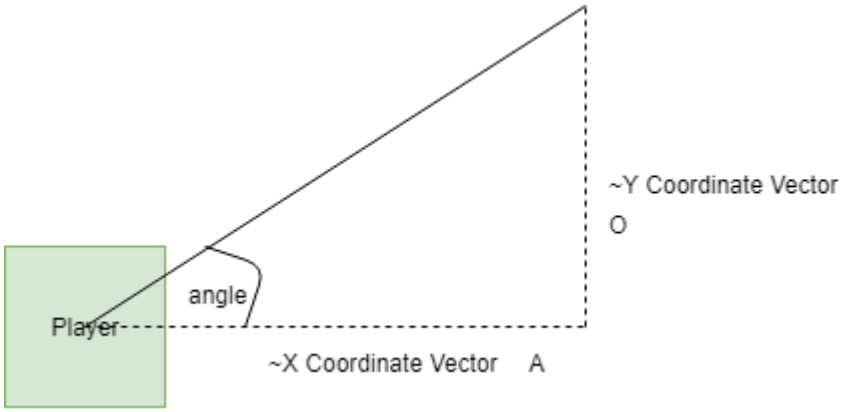
Movement Design

The character will be moved in all four directions with WASD buttons. This allows the player to move wherever they want, but also diagonally when two keys are pressed at the same time.
The mouse will also be used as a key aspect of the movement system to all the user to spin 360 degrees to enable them to shoot in the direction they want.

Diagram of how the 360 aiming system will work



To rotate the player I will need to use the pygame command: pygame.transform.rotate(), for this command to work, you will need to give it two parameters. Number 1 – What to act on, in this case, the player's image. Number 2 – the angle that you want the image to be rotated by. Despite this, math can be used to find out the angle between the player and the aim line.



Real world equations

To Work out the angle, you need to find the difference from the points in the center of the sprite to the points where the mouse is positioned. With this, you can work out the angle with trigonometry by doing; tan-1(O/A) and this will give you the angle of the aim line in degrees.

Pygame rotate source https://www.pygame.org/docs/ref/transform.html#pygame.transform.rotate

Math atan2 function source https://www.geeksforgeeks.org/atan2-function-python/

Pseudocode

```
MOUSEX, MOUSEY = PYGAME.MOUSE.GET_POS()
CENTERX, CENTERY = GET.RECT.CENTER
XVECTOR = MOUSEX – CENTERX
YVECTOR = MOUSEY – CENTERY
ANGLE = MATH.ATAN2(XVECTOR, YVECTOR)
ANGLE = MATH.DEGREES(ANGLE)
PYGAME.TRANSFORM.ROTATE(PLAYERIMAGE, ANGLE)
```

Pseudocode Comments

The function pygame.mouse.get_pos will return both x and y coordinate of the mouse's current position, this will be the first point in the vector. The next function, get.rect.center will return the centre x and y coordinates of the player, this will be the second point in the vector. The next two lines of code will create a vector for the x and y plane by subtracting the x and y coordinate from both points. After this, the atan2 function located within the math library will be used to find the angle in radians from the two vector points. The next line of code will then convert radians to degrees; this allows the command pygame.transfrom.rotate to rotate the sprite in the direction facing the vector.

Collision Detection

The player sprite will also need collision detection set up, so the player is unable to no clip through the walls and get stuck in the room. The player will also need collision detection between sprite groups; during my research this doesn't seem to be a hard task to program as the library pygame has a built-in function that checks collisions between sprite groups. Despite this, the player sprite is going to need to collide with the wall sprites within the room a different way by moving along the side in all four direction.

Sprite collide source https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.spritecollide

Pseudocode

```
Collision_list = pygame.sprite.spritecollide(player, walls, False)
For wall_collided in collision_list:
        If velocityx > 0:
                Player.rect.right = wall_collided.rect.left
        else:
                Player.rect.left = wall_collided.rect.right
For wall_collided in collision_list:
        if velocityy > 0:
                 player.rect.bottom = wall_collided.rect.top
        else:
                 player.rect.top = wall_collided.rect.bottom
```

Pseudocode Comments

The code starts off with a list that is always empty if the player sprite is not touching a wall in the other sprite group. The FOR-loops check on both the x and y plane to see when something is colliding with the player, what orientation their both in. Then it does a check to see if the velocity is greater than zero, this then shows what direction the player is going; if the velocity is positive in the x, the player is moving right. If the velocity is negative, the player is moving left; this applies to the y plane aswell. The next line of code will keep the player side to the exact coordinates of the wall's opposite side to make sure there is no way out of the map.

Boss Ideas/Design

At the end level of the game, the player will be matched up against a boss. This boss will have a lot of health and several attacks to perform when the player is battling against it; the boss level will by far be the hardest level within the game to really challenge the user.

Boss #1

This boss with shoot out 3 projectiles, which the user will have to dodge. This attack can be performed by the boss with an interval of time. Despite this, he will have increased health but low attack damage. When one of the bosses' projectiles hits the player, the player will lose health.

Above is a rough layout of what the boss level for boss #1 will look like, this will be the sixth room after the user has proceeded through the previous five rooms.

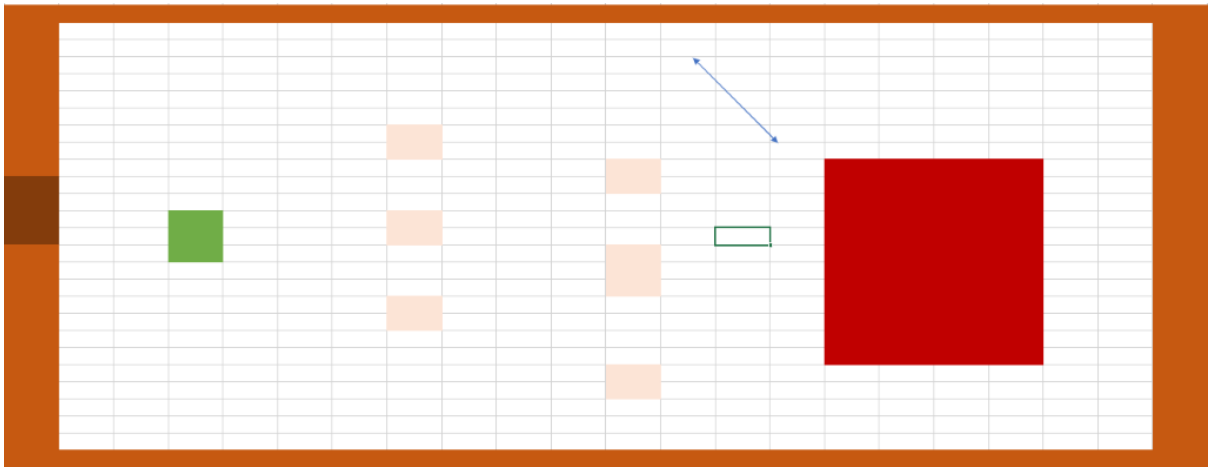
Boss #2

This boss has low attack damage, when in this form, but once all of its health has been depleted the boss will split into three mini-bosses which will be slightly faster and have half the original boss's health each. In order to complete this boss level, the player must beat both forms of this boss. For this boss to take HP off the main character, the player must not get hit by the boss when its moving around.

When the HP of the boss above is zero. The boss will split into 3 smaller bosses, as seen below.

Boss #3

This boss will move around the room, bouncing off the walls and it will also shoot several projectiles at you in a reoccurring time interval. The player will need to dodge the projectiles as well as the boss moving around the room. With the boss moving at a medium velocity, if you're not paying attention; the hit from the boss will take a lot of health. Despite this, the projectiles will not do as much damage as the boss hit but will still do more damage to the player's HP than a regular enemy attack.



To have the boss come at the wall at a 45-degree angle, the boss should bounce of the wall at another 45-degree angle in the opposite direction, so it doesn't get stuck between two points bouncing back and forth.

Pseudocode

REPEATED

```
direction = random.randint(1, 4)
if direction == 1:
        vx, vy = neg_velo, neg_velo
 if direction == 2:
        vx, vy = pos_velo, pos_velo
```

```
if direction == 3:
        vx, vy = neg_velo, pos_velo
 if direction == 4:
        vx, vy = pos_velo, neg_velo

vx *= -1
vy *= -1
boss.x += vx
boss.y += vy

if rect.x == 64 or rect.x == (WIDTH - 256):
      if direction == 1:
            direction = 4
      elif direction == 2:
            direction = 3
       elif direction == 3:
            direction = 2
       elif direction == 4:
            direction = 1
 if rect.y == 64 or rect.y == (HEIGHT - 256):
       if direction == 1:
            direction = 3
       elif direction == 2:
            direction = 4
       elif direction == 3:
            direction = 1
       elif direction == 4:
            direction = 2
```

REPEATED

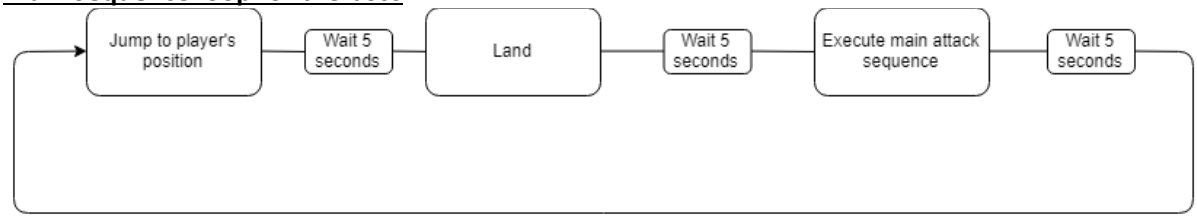
Pseudocode Comments

The code will firstly generate a random number between 1-4, this variable will be picked up by the if statements, if one is true it will change the boss' x and y velocity respectively. Next the code will multiply the velocity integer by –1 to flip the directions; then the program will proceed to update the boss' x and y coordinate. This will keep happening until the boss makes either parameter in the IF statement true. Once an indented IF statement becomes true the direction of the boss will change to the opposite direction and repeat: the code will repeat due to the boss' update function being located in a higher object class, in which will be located under a while loop.

Boss #4

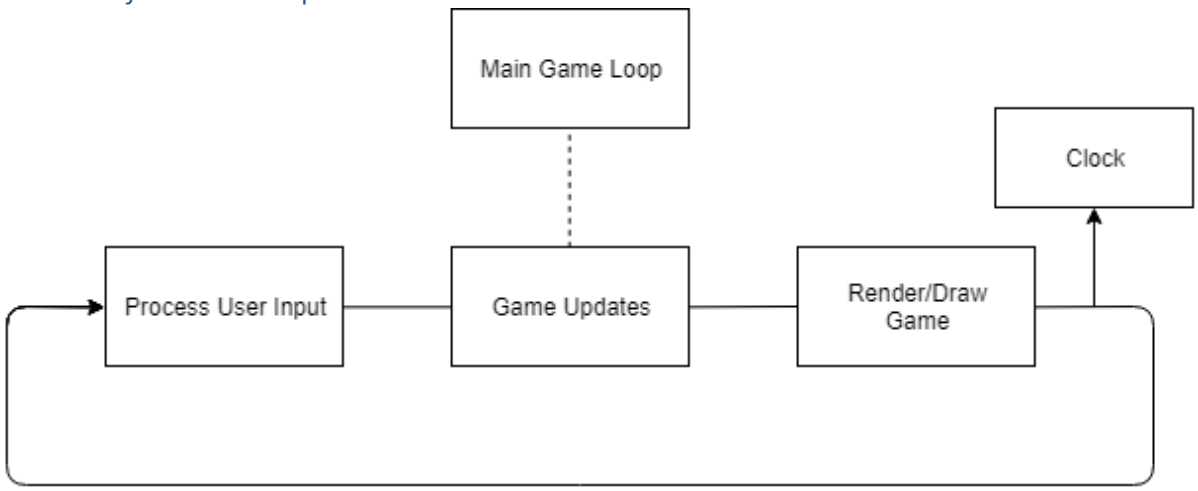
This boss will be able to jump around within the map as part of its attacks; for 5 seconds there will be a shadow masked on the floor which will show the position of where to boss will land. But once landed, it's immobile. Furthermore, the boss will track the player's position when it starts its jump attack, and later jump to that position, this causes the player to have to move from under the boss, otherwise a large amount of damage will be dealt to the player. Once the boss has landed there will be a 10 second window to attack the boss before the main attack commences: The main attack is the boss letting out 8 projectiles every 45 degree around him. The player will need to dodge these projectiles. Despite this, the projectiles only disappear when the player is hit, or a wall entity is hit.

Main Sequence loop for the boss



The boss will follow this sequence until its health is at zero or the player's health is at zero. Despite this, the 5 second intervals give the player a chance to react to these attacks, so the boss is not extremely difficult to beat.

Processes of the Main Loop of the Game



-This shows how the game will run on its main loop to make sure everything is processed, updated then rendered back onto the game window.

Process User Input – This is a collection of all the keystrokes that the user presses while playing the game, the program should recognise this and execute a part of code that's is relevant to that input.

Game Updates – Once the input has been received, the game's code should run and change through a process of selection to figure out what the player should do, then execute it. In the python library 'pygame' you are able to see the events by using the command 'pygame.event.get()'.

Pseudocode

```
Loop = True
WHILE Loop == True
    FOR event in pygame.event.get()        # Loops over every movement that the ser makes.
        PRINT event
        IF event.type == pygame.QUIT()  # '.type' breaks down the events into types so
they                    Loop = False                    can be compared. This type is also the red 'X'
in                                                  the top right corner.
        END IF
    END FOR
END WHILE
Pygame.quit() # Exits pygame and closes the window
```

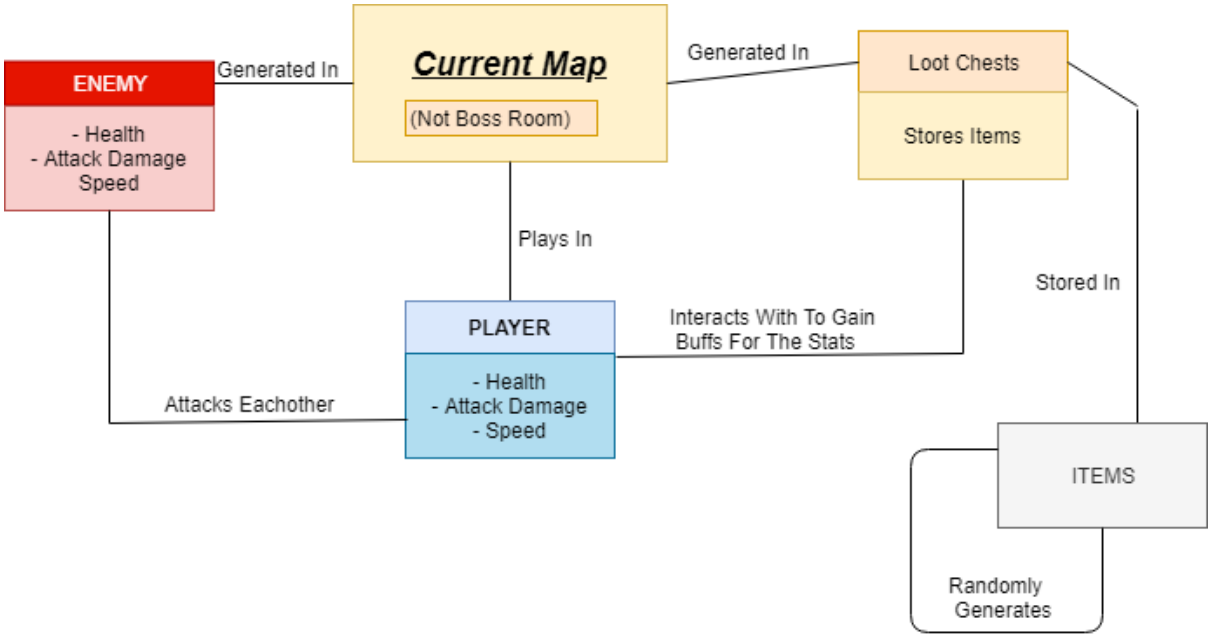
Render/Draw T he Game – This is the final stage of the loop; once all of the inputs and updates have been done the game now needs display and draw everything onto the screen and make sure it matches the movement in the update part of the code.
e.g - when the user presses 'W' -- move player up 20 pixels. The on-screen player must move up 20 pixels.

Clock – In basic terms the clock is the amount of time taken for this loop to be processed. Despite this, every computer will have different processing times to process these stages, this resembles the FPS(frames per second) in the game, this brings up the problem of people with faster computers running the game too fast which could give them an advantage in the game compared to people with a slow computer. To fix this, the clock speed can be capped so all computers run the game at the same speed.

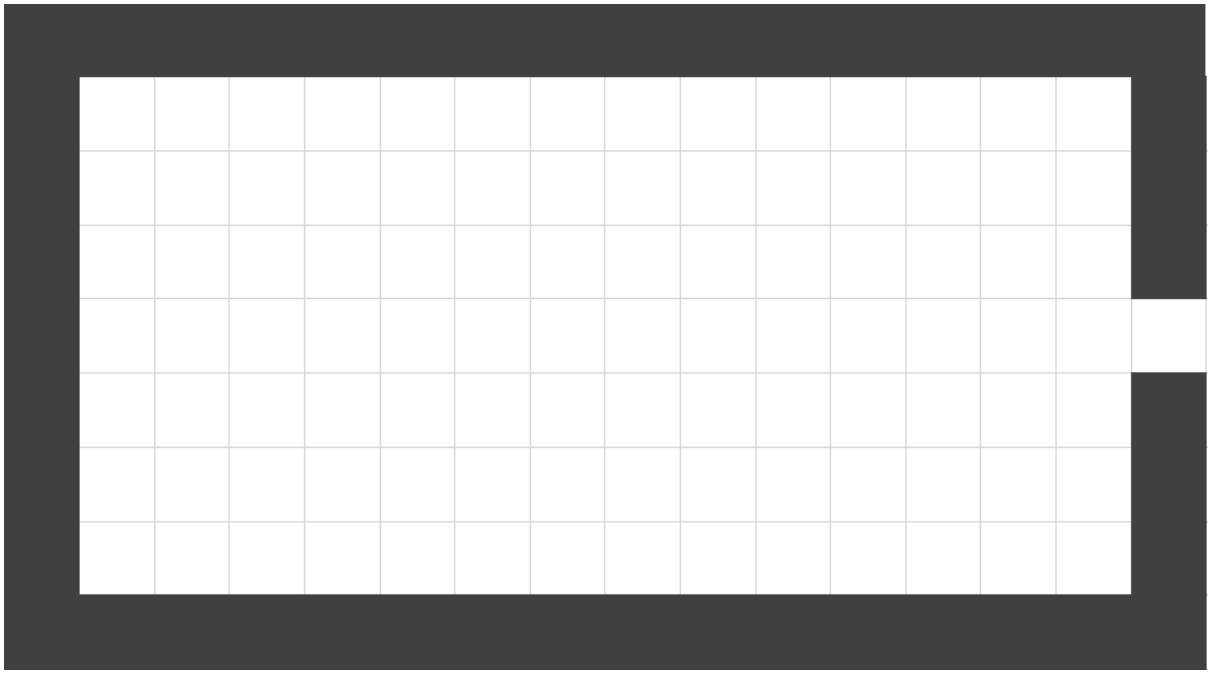
Map Design

The map will be a big part of the program, because this is what the player will be spending most of their time interacting with objects within the room.
The Map needs to randomly generated each time the player plays the game, to offer a different gameplay experience each time. The map needs to allow the player have access to the door(s) in the room, so they're not blocked by an obstacle that the player cannot pass through, therefore restricting the game to carry on.
The Map will also be surrounded by an outer blockade to make sure that the player is unable to get out of the room and go off screen. With this outer edge barricade, it will be accompanied by door(s) so the player is able to exit and enter new rooms. If there are enemies present in the rooms, the doors will remain closed, not allowing the player to get through, until they have all been killed, in which, they will open.

Entity Relationship Diagram



-The diagram above shows how the entities in the non-boss room interact with each other.



Depending on the room, the room will, for the most part, look like the diagram above with different generated objects initialised within that room.

Pseudocode

```
FOR I in range(GRIDNUMBER):
      Create_wall(GRIDWIDTH, 0, wall_image)
FOR k in range(GRIDNUMBER):
       Create_wall = Wall(GRIDWIDTH,(HEIGHT - 64), image)
FOR u in range(1, GRIDHEIGHT):
       Create_wall = Wall(0,GRIDHEIGHT ,image)
FOR o in range(1, GRIDHEIGHT):
```

```
        Create_wall = Wall((WIDTH - 64), GRIDHEIGHT, image))
    Wall.add(spritegroup)
```

Pseudocode Comments

The code above just essentially will loop through a FOR loop and place a wall along the top, bottom and left/right side of the map. This will act as a "boundary" so the player sprite will be unable to escape the map when they collide with it. This bit of code will be run on every room of the game.

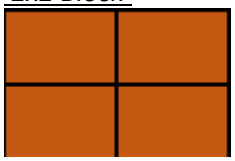
Random generation

To have the map randomly generating structures within the room can be done by utilising the random library.

The objects that are going to be randomly placed into the map can be seen as a certain 'blocktype', each block type will be different with the randomness that you can have any one placed around the room with several different combinations.

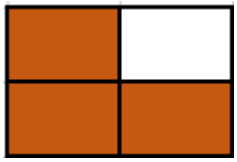
Block types:

'2x2 Block'

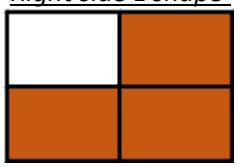


This will most likely be the largest block type, as there's not a lot of room to work with, due to the grid sizing being large. This is because errors can arise when working with larger block types not allowing enough room for other structures in the room

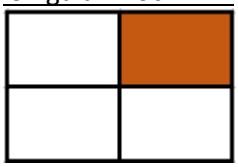
'Left side L shape'



'*Right side L shape'*



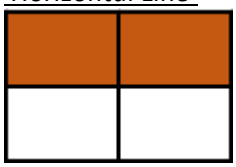
'Singular Block 1x1'



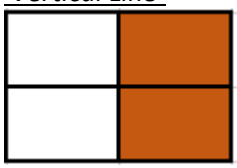
'Left Side r shape'

'Right Side r shape'

'Horizontal Line'



'Vertical Line'



Pseudocode

```
MAP_LIST = []
FOR I in range(WALL_AMOUT):
...
        Validwallspot = Fasle
         While Validwallspot = Fasle:
                BLOCK_TYPE = RANDOM_BLOCK
                XVector = RANDOM_X_NUMBER
                YVector = RANDOM_Y_NUMBER
                IF BLOCK_TYPE = BLOCK_TYPE_NUMBER:
                        IF MAP_LIST[XVector][YVector] = " ":
                                MAP_LIST[XVector][YVector ] == WALL
                                ...
                                Create_wall = WALL(XVector, YVector)
                                Wall.add(spritegroup)
                                Validwallspot = True
```

Pseudocode Comments

The code above shows what the random generation algorithm looks like. In brief, the program loops around a certain amount of times in a FOR loop, this is followed by a boolean variable which acts as a check to see if the spot is valid on the map. The while loop makes sure it loops the code under it until a valid spot has been found.
With this, the BLOCK_TYPE variable will pick, at random, a block type from the diagrams above and check that block type against a random x and y vector to see if the space is available.
If the space is available, the code will now mark that space in the list as occupied, to update for any other walls that need to be added, so they do not generate over each other.
After this, the wall object will be called with its x and y vector parameters passed through it.
Then, the valid spot boolean check is set to true so the FOR loop can proceed to the next block.

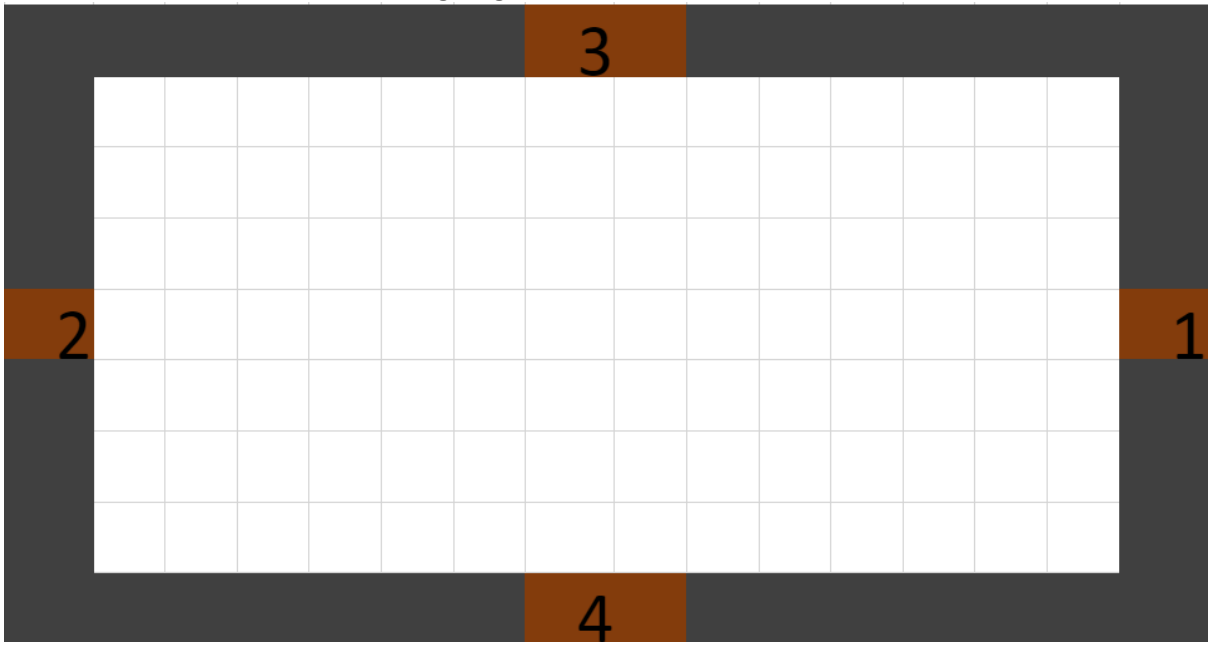
The code in the actual program will be longer due to there being 8 block types and each one needs to have an IF detection statement whilst making sure each block of the blocktpye is marked as occupied in the list.

Map Door Placement

When starting out, there will only be one door in the starting room to lead to other the next room. When coding these doors in there should be no repeated directions, so it doesn't get confusing for the client. Doors should also not be accessible when there are still enemies within the room; so the doors must be closed and act as a collision when enemies are present. Each door will have its own collision detection with the player and will be the indicator either pointing towards the previous

room or the next room on, so the program can know where the player is throughout the map and update/draw that current scenario at that current point.

Each door will have its own variable giving it its own direction.



When loading to each room, the player will not actually be loaded into a room, the room itself will load around the player, with this I will need to create the illusion of the player spawning on the other side of the door, walking into a new room.

Pseudocode

```
if doordirection == 1:
        player.x = (64)
        player.y = (352)
 elif doordirection == 2:
        player.x = (WIDTH - 144)
        player.y = (352)
elif doordirection == 3:
        player.x = (512 - 32)
        player.y = (HEIGHT - 144)
  elif doordirection == 4:
          player.x = (512 - 32)
          player.y = (64)
```

Pseudocode Comments

This code will look at the direction of the door that the player is coming through, when collided with that door, the detection will tell this piece of code to run and change the player's x and y coordinates to a specific point to make it look like the player is coming out the door when the room just loaded around them.

Enemy Design

The enemies will play a big part in the non-boss rooms when playing through the game; they will be chasing the player sprite down with a pathfinding algorithm. If the player is hit by one of these sprites, they player's health will lower. However, if one of these enemies gets shot by one of the player's projectiles, their health will also lower.
There are going to be 3(min)-6(max) enemies spawning in a room at once with a random amount and a random placement.

Each enemy will be 64x64 and not be able to pass through walls/doors to get to the player sprite. The enemy object will need a function to calculate what node it resides, in order to properly path find and stay within the boundary of the map.

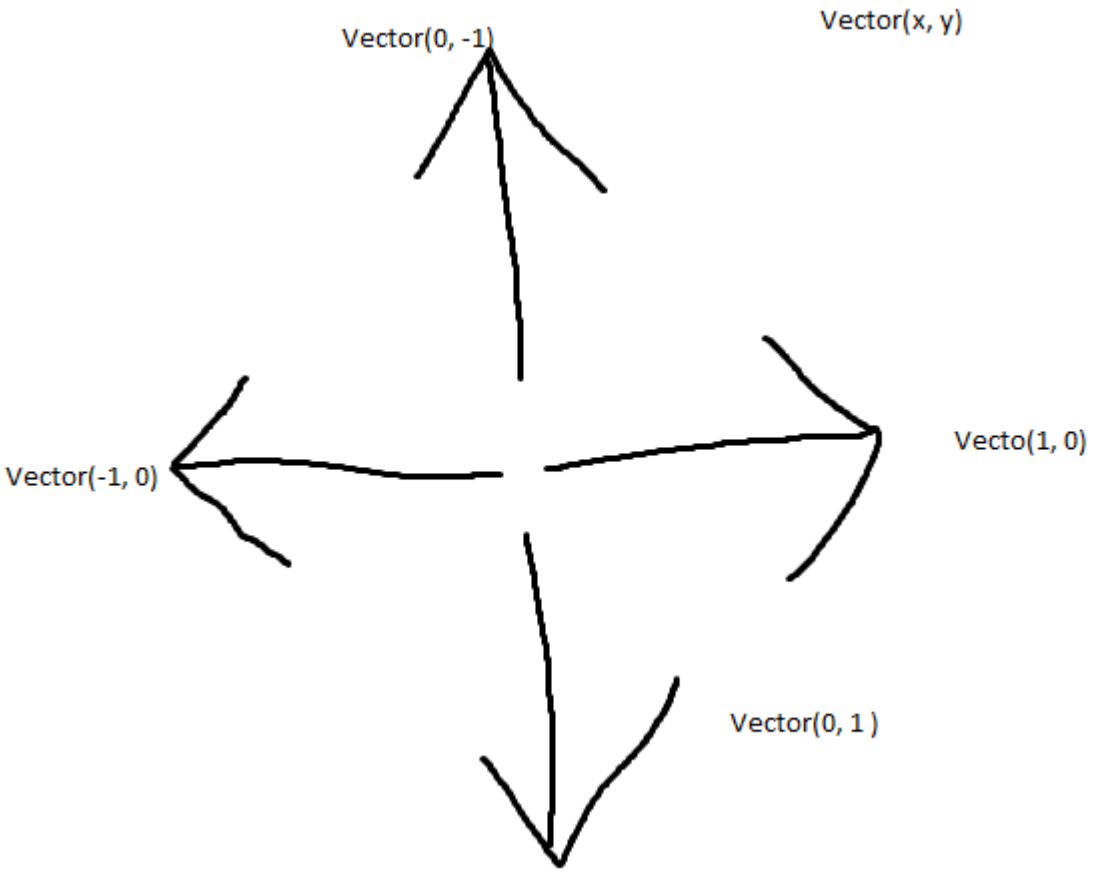
Pseudocode

```
iny = False
inx = False
 for i in range(0, WIDTH, TILESIZE):
        if i == enemy.x and (i + 64) == (enemy.x + 64) :
           coordx = i
           inx = True

     for j in range(0, HEIGHT, TILESIZE):
        if j == enemy.y and (j + 64) == (enemy.y + 64) :
           coordy = j
           iny = True

       if iny == True and inx == True:

              x = int(coordx / 64)
              y = int(scoordy / 64)

               return (x, y)
```

Pseudocode Comments

The code in the function starts off by setting both boolean checks to false. After this, a FOR loop is used from the coordinates 0 to 1024(WIDTH) of the screen, with a step of 64(TILESIZE). So, all this will do is step from 0 to 1024 in multiples of 64. There are two FOR loops to check both the x and y nodes separately. The IF statements will look to see if the x and y coordinates are equal to I or j. if this is returns true, the variable coordx will be set as I and the boolean value in(x,y) will be set to true. Furthermore, the function will only return the x and y node value if both nodes are valid and have been located.

When the enemy needs to path find for the shortest path possible several algorithms can be used, but before all this the map must be set up in a node format, this can be done using vectors to allow the pathfinding algorithms to easily identify a node and the location of it.
When pathfinding the program will look at the current node and look at the nodes all around itself, these are generally called "node neighbours" and these node neighbours indicate what the next node the player can travel to. These node neighbours are represented by directional vectors.



Breadth First Search

The enemies will need to chase the player sprite with pathfinding algorithm but item #2 will utilize the enemy's pathfinding by using breadth first search. I'm going to use this search due the item remaining at a static point and not moving; this will make sure the program will not run into no performance issues due to the path being direct and not changing constantly.

Pseudocode

```
Parameters = map, start_node, goal_node
    list = queue
    list.append(start_node)
    path = {]
    while len(list) > 0:
      Current_node = list.popleft()
      if current_node == goal_node:
        break
      if map.node_neighbours(current_node) != None:
        for next_node in map.node_neighbours(current_node):
          if next_node not in path:
            list.append(next_node)
            path[next_node] = current_node – next_node
    return path
```

Pseudocode Comments

The code starts with a queue then the start node is appended to it. After this, the path dictionary is created. The next line of code checks to see if the length of the list is greater than zero. If true, the variable current_node will be assigned the value in the left most section of the list, while deleting it at the same time with the pop command. The next part of the code is a check to see if the

current node that the sprite is located in, is equal to the goal node, if true, the function will experience a break function and the rest of the function will no run, this will be implemented due to performance issues just in case the program Is hindered in any way. Furthermore, an IF statement will be used to see if the current node has any neighbouring node, if true, a FOR loop will occur to search through all the current node's neighbours. The next If statement will see if the node is in the path dictionary, if not, the program will proceed to append it to the list and create a directional vector pointing towards the next node. Once the list length is equal to zero the program will return the path.

A star Search

Furthermore, the enemies will use A star path finding algorithm to find the shortest path to chase the player down; this type of pathfinding will be used as it's the most efficient and will not put much performance strain on the game.

Pseudocode

```
Parameters = map, start_node, goal_node
List = PriorityList
List[0] = start_node
Path_cost = {}
Path = {}
While len(List) != 0:
        current_node = list[1]
        If current_node == goal_node:
                Break
        For next_node in map.node_neighbour(current_node):
                next_node_cost = path_cost[current_node] + map.cost(current_node, next_node)
                If next_node not in Path_cost or next_node_cost < cost[next_node]
                        node_priority = next_node_cost +heuristic_estimate(goal_node,next_node)
                        List.put(next_node, node_priority)
                        Path[next_node] = current_node – next_node
return Path
```

Pseudocode Comments

The code will start off by generating a priority list to see what nodes take priority over eachother. The code will them assign the enemies start node to the start of the priority list. Despite this, two dictionaries will be created to relate the nodes with the path as the key and the cost of transport with the value. The code will then go into a WHILE loop which checks to see if the priority list is empty, if false, the program will return the path as empty; indicating the movement from the start and goal node has been complete. If true, the program will assign the second item in the list as the current node. The code will then go through an IF statement to see if the goal node has been reached, if true, the program will run the break command in python, which will be used for performance, due to the subroutine not having to run if this command is executed. If false, a FOR loop will be used to look at all the neighbouring nodes of the current node that the player is currently located in. The next line will then calculate the cost of these neighbouring nodes. The next IF statement then checks to see if the cost of that node is stored in the cost dictionary or to see if the next node cost is lower than the next node. Despite this, the program will use the next node cost and add it to an estimate heuristic value from the goal node to the next node. This will then get placed into the priority list and then the current node will be subtracted from the next node to give a directional vector for the sprite to follow. If the list length is zero, the function will return the full path with what node the sprites should take.

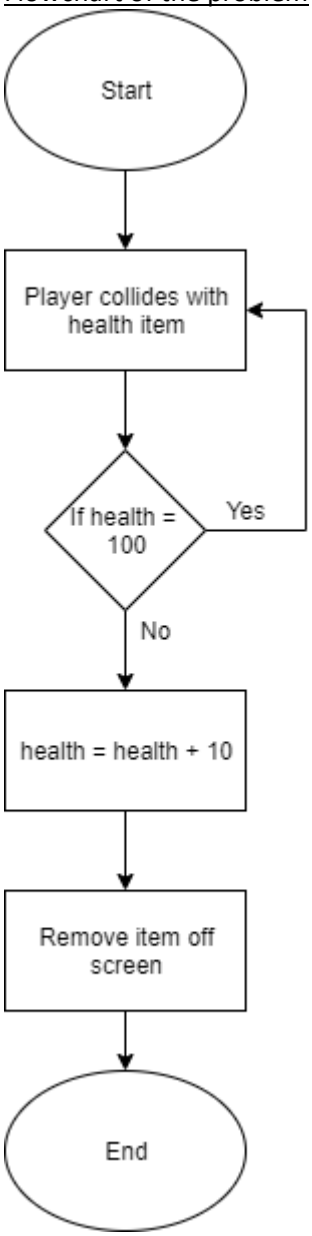
Item Design

Items within the level will help the player along their progression through the rooms and give them an extra boost. The items will be randomly generated into each room, so the player does not have a clear-cut outcome and will not be able to predict what each room is going to output.

Item #1

This item will take the form of a health pack and regenerate the player +10 health when the player interacts with it. The health pack will not give any extra health to the player once the max of 100 health has been reached. I must make sure the program does this to ensure that there are no errors that come up when the program runs and when the player collides with this item, the item must not be used and disappear, but can be kept in that current room for a later date if the player needs to go back to it.

Flowchart of the problem



Item #2

This item will act as a 'taunt', drawing the enemies in towards it so the player can free rein of the room while the enemies are all surrounded by one point. This item will have a low chance of spawning, due to it being a lot more over-powered. This item will use the path-finding algorithm breadth-first search to draw all enemies towards it. As talked about in my project analysis I feel breadth-first search could cause performance problems when an enemy to player path finds due to the constant dynamic changing nodes. Despite this, I feel this algorithm would be correct for a static node as the algorithm does not have to work as hard to recalculate the path each time, my client also agrees that this would be a good addition into the game.

Libraries

The main library that I will have to use is pygame; this will be the main basis of my game that I am creating. Pygame will help with the sprites, allowing useful commands to be used when working with sprites. But, pygame also offers more useful commands with using hitboxes within the .rect commands.
The library time will also aid in the help of animating the sprites to make sure everything is timed with the set number of slides for the animation to take place.
The random library will also play a big part in the code by leaving a lot of aspects to chance. E.g - room layout, enemy placement and boos type. This allows the game to be different every time you play it, due to the randomness aspect that occurs when picking and choosing what objects randomly.
The math library will be used to aid in mathematical equations that the program will run to make the program more efficient. The library will allow the built-in functions to easily execute, while not hindering the performance of the game.

Annotated Code

```
1.  import pygame as pg
2.  import sys
3.  import math
4.  from os import path
5.  import random
6.  from collections import *
7.  import heapq
8.
9.  # // Libary imports //
10.
11. WIDTH = 1024
12. HEIGHT = 768
13. vector = pg.math.Vector2
14.
15. TILESIZE = 64
16.
17. GRIDWIDTH = WIDTH / TILESIZE
18. GRIDHEIGHT = HEIGHT / TILESIZE
19.
20. PLAYERVELOCITY = 5
21. # // Globals //
22.
23.
24. class PriorityQueue:
25.     # Priority queue initialize
26.     def __init__(self):
27.         self.nodes = []
28.     def put(self, node, cost):
29.         # Put an item into the Queue
30.         heapq.heappush(self.nodes, (cost, node))
```

```
31.     def get(self):
32.         # Return the item in slot one and pop it
33.         return heapq.heappop(self.nodes)[1]
34.     def empty(self):
35.         # Returns true of false
36.         return len(self.nodes) == 0
37. 
38. class Wall(pg.sprite.Sprite):
39.     # Initialize wall with x and y coordinate and the image
40.     def __init__(self, x, y, image):
41.         pg.sprite.Sprite.__init__(self)
42.         self.image = image
43.         self.rect = pg.Rect(x, y, 64, 64)
44.         self.x = x
45.         self.y = y
46. 
47. class Door(pg.sprite.Sprite):
48.     def __init__(self, x, y, image, doordirection):
49.         pg.sprite.Sprite.__init__(self)
50.         self.image = image
51.         #  Makes sure that the door in placed on the
52.         #  outer edge with
53.         #  the right hitbox for the four different directions.
54.         if doordirection == 1 or doordirection == 2:
55.             self.rect = pg.Rect(x, y, 32, 128)
56.         if doordirection == 3 or doordirection == 4:
57.             self.rect = pg.Rect(x, y, 128, 32)
58.         self.x = x
59.         self.y = y
60. 
61. class ClosedDoor(pg.sprite.Sprite):
62.     def __init__(self, x, y, image, doordirection):
63.         pg.sprite.Sprite.__init__(self)
64.         self.image = image
65.         # Makes sure that the door in placed on the
66.         # outer edge with
67.         # the right hitbox for the four different directions.
68.         if doordirection == 1 or doordirection == 2:
69.             self.rect = pg.Rect(x, y, 32, 128)
70.         if doordirection == 3 or doordirection == 4:
71.             self.rect = pg.Rect(x, y, 128, 16)
72.         self.x = x
73.         self.y = y
74. 
75. 
76. class Projectile(pg.sprite.Sprite):
77.     def __init__(self, startposx, startposy, endposx, endposy, velocity, screen, pr
    ojtype):
78.         pg.sprite.Sprite.__init__(self)
79.         # Checks what type of projectile
80.         if projtype == 1:
81.             self.image = pg.image.load("Projectile.png")
82.         if projtype == 2:
83.             self.image = pg.image.load("orb_red.png")
84.         self.rect = self.image.get_rect()
85.         self.x = startposx
86.         self.y = startposy
87.         self.endx = endposx
88.         self.endy = endposy
89.         self.screen = screen
90.         # Create x and y vectors for projectiles
91.         self.difx = self.x - self.endx
92.         self.dify = self.y - self.endy
93.         self.vel = velocity
94. 
95.     def update(self):
```

```
96.         # Calls the off screen check which just checks if the hitbox of
97.         # the projectiles go off screen, if so, they are terminated
98.         if self.OffScreencheck() == True:
99.             self.kill()
100.                 print("Killed")
101.             # Creating the hypotenuse vector equalling to the distance
102.             self.dist = math.hypot(self.difx, self.dify)
103.             try:
104.                 # Calculate the velocity
105.                 self.vx = (self.difx / self.dist) * self.vel
106.                 self.vy = (self.dify / self.dist) * self.vel
107.                 # Move the projectile taking away the velocity each time
108.                 self.x -= self.vx
109.                 self.y -= self.vy
110.                 self.rect.center = (self.x, self.y)
111.             except ZeroDivisionError:
112.                 #Except statment to catch a zero
113.                 #error in the case that
114.                 #self.distance is equal to zero.
115.                 self.kill()
116.         def OffScreencheck(self):
117.             # Check to see if the projectile is off screen
118.             if (self.rect[0] + 16) < 0 or self.rect[0] > WIDTH or (self.rect[1]
    + 16) < 0 or self.rect[1] > HEIGHT:
119.                 return True
120.             else:
121.                 return
122.
123.     class Enemy(pg.sprite.Sprite):
124.         def __init__(self, map, startposx, startposy, health = 60):
125.             pg.sprite.Sprite.__init__(self)
126.             self.imgindex = []
127.             # Initalizing all images
128.             self.imgindex.append(pg.image.load("emain1.png").convert_alpha())

129.             self.imgindex.append(pg.image.load("emain2.png").convert_alpha())
130.             self.imgindex.append(pg.image.load("emain3.png").convert_alpha())
131.             self.imgindex.append(pg.image.load("emain4.png").convert_alpha())

132.             self.imgindex.append(pg.image.load("emain5.png").convert_alpha())

133.             self.imgindex.append(pg.image.load("emain6.png").convert_alpha())

134.             self.imgindex.append(pg.image.load("emain7.png").convert_alpha())

135.             self.imgindex.append(pg.image.load("emain8.png").convert_alpha())


136.
137.             self.vx, self.vy = 0, 0
138.             self.index = 0
139.             self.room = map
140.             self.image = self.imgindex[self.index]
141.             self.timer = 0
142.             self.rect = self.image.get_rect()
143.             self.rect.x = startposx
144.             self.health = health
145.             self.rect.y = startposy
146.             self.center = self.rect.center
147.             # Setting the speed of the enemy
148.             self.enemy_speed = 4
149.             self.hitbox_rect = pg.Rect(startposx, startposy, 60,60)
150.
151.         def getCenter(self):
152.             # Returns enemy sprites center
153.             self.center = self.rect.center
154.             return self.center
```

```
155.
156.           def update(self, proj_group, playercenter, node, wall_group, itemnode, s
    pecialitem):
157.                self.goal = node
158.                # Checks to see if the enemies health is at zero, then
159.                # the object is deleted with the .kill() command
160.                if self.health == 0:
161.                    self.kill()
162.                    return True
163.                # Returns the node that the enemy is in
164.                self.currentnode = self.get_node()
165.
166.                if itemnode == 1:
167.                    # Breadth first search
168.                    self.path = breadthfirst_search(self.room,specialitem, self.curr
    entnode)
169.                else:
170.                    # A star search
171.                    self.path = aStar(self.room,self.goal, self.currentnode)
172.                if self.currentnode != False:
173.
174.                    # Checks to see if the current node is on the goal node
175.                    if self.currentnode != self.goal:
176.                        try:
177.                            # Add the current node to the next node in the list.
178.                            self.current = self.currentnode + self.path[(self.curren
    tnode)]
179.                        except:
180.                            # Do nothing
181.                            pass
182.                        # Pass the current node into a back-up node for other
183.                        # iteratons
184.                        self.backup = self.currentnode
185.                else:
186.                    # Checks to see if the current node is the goal node
187.                    if self.currentnode != self.goal:
188.                        try:
189.                            self.current = self.backup + self.path[(self.backup)]
190.                        except:
191.                            pass
192.
193.                if self.currentnode == False:
194.                    pass
195.                else:
196.                    # Works out the directional vector
197.                    self.dif = self.currentnode - self.current
198.
199.                # Checking what direction the next path is
200.                if self.dif == [1, 0]:
201.                    self.rect.x -= self.enemy_speed
202.                elif self.dif == [-1, 0]:
203.                    self.rect.x += self.enemy_speed
204.                elif self.dif == [0, 1]:
205.                    self.rect.y -= self.enemy_speed
206.                elif self.dif == [0, -1]:
207.                    self.rect.y += self.enemy_speed
208.                # Checks to see if the enemy gets hit by a projectile
209.                # and takes away 10 health
210.                if pg.sprite.spritecollide(self, proj_group, True):
211.                    self.health = self.health - 10
212.                self.rect = self.hitbox_rect
213.                self.playercenter = playercenter
214.                # Rotate towards player procedure
215.                self.rotatetowardsPlayer()
216.                self.random_timer = random.randint(2, 5)
217.                self.timer += self.random_timer
```

```
                # This is where you a able to adjust the
                # animation time, for the cycles of the pictures.
                if (self.timer % 20) == 0:
                    self.index += 1
                    if self.index >= len(self.imgindex):
                        self.index = 0
                    self.image = self.imgindex[self.index]

            # Rotates the enemy to look towards the player
            def rotatetowardsPlayer(self):
                self.angle_vec = math.atan2((self.center[0] - self.playercenter[0]), (self.center[1] - self.playercenter[1]))
                # The angle is converted from radians to degrees
                self.angle = math.degrees(self.angle_vec)
                # Rotates the image about an angle
                self.newimage = pg.transform.rotate(self.image, self.angle - 180)
                oldcenter = self.rect.center
                self.newrect = self.newimage.get_rect()
                self.newrect.center = oldcenter

            def get_node(self):
                # Need to be set to False again for
                # when the procedure is called again in the update section
                self.iny = False
                self.inx = False
                self.playercenter = self.getCenter()
                for i in range(0, WIDTH, TILESIZE):
                    # Checking if the player's center "x"
                    # coordinate is between a tile's area coordinates
                    if i == self.rect.x and (i + 64) == (self.rect.x + 64) :
                        self.coordx = i
                        self.inx = True

                for j in range(0, HEIGHT, TILESIZE):
                    if j == self.rect.y and (j + 64) == (self.rect.y + 64) :
                        self.coordy = j
                        self.iny = True
                # Searching through the tile list and
                # mapping out what tile the player's center is in
                if self.iny == True and self.inx == True:
                    # dividing the x and y coordinates
                    # by 64 and minusing 1 to get into list form
                    x = int(self.coordx / 64)
                    y = int(self.coordy / 64)

                    return (x, y)
                return False

            def draw(self):
                self.screen = maingame.returnGameScreen()
                if self.health == 0:
                    return True
                # Only draws to the screen if the health is 0
                else:
                    try:
                        self.screen.blit(self.newimage, self.newrect)
                    except:
                        self.screen.blit(self.image, self.rect)

        class Item(pg.sprite.Sprite):
            def __init__(self, startposx, startposy, item_number):
                # initalizing item sprite
                pg.sprite.Sprite.__init__(self)
                # Assigning items to the correct image
                if item_number == 1:
                    self.image = pg.image.load("item1.png")
```

```
283.                 if item_number == 2:
284.                     self.image = pg.image.load("orb_green.png")
285.                 self.rect = pg.Rect((startposx + 16), (startposy + 16), 32, 32)
286.                 self.x = startposx
287.                 self.y = startposy
288.
289.         class PlayerSprite(pg.sprite.Sprite):
290.             def __init__(self, startposx, startposy, health = 100):
291.                 pg.sprite.Sprite.__init__(self)
292.
293.                 self.imgindex = []
294.                 # Loading all animation images
295.                 self.imgindex.append(pg.image.load("main1.png").convert_alpha())
296.                 self.imgindex.append(pg.image.load("main2.png").convert_alpha())
297.                 self.imgindex.append(pg.image.load("main3.png").convert_alpha())
298.                 self.imgindex.append(pg.image.load("main4.png").convert_alpha())
299.                 self.imgindex.append(pg.image.load("main5.png").convert_alpha())
300.                 self.imgindex.append(pg.image.load("main6.png").convert_alpha())
301.                 self.imgindex.append(pg.image.load("main7.png").convert_alpha())
302.                 self.imgindex.append(pg.image.load("main8.png").convert_alpha())
303.                 self.index = 0
304.                 self.health = health
305.                 self.image = self.imgindex[self.index]
306.                 self.timer = 0
307.                 self.pos = vector(startposx, startposy) * TILESIZE
308.                 self.rect = self.image.get_rect()
309.                 self.center = self.rect.center
310.                 self.hitbox_rect = pg.Rect(startposx, startposy, 60,60)
311.                 self.original_image = self.image.copy()
312.                 self.vx, self.vy = 0, 0
313.                 self.rect.x = startposx
314.                 self.rect.y = startposy
315.                 self.hittimer = 0
316.                 self.boss_hittimer = 0
317.
318.             def keys(self):
319.                 # Key logging what direction the player is moving
320.                 self.vx, self.vy = 0, 0
321.                 keys = pg.key.get_pressed()
322.                 if keys[pg.K_w]:
323.                     self.vy = -PLAYERVELOCITY
324.                 if keys[pg.K_a]:
325.                     self.vx = -PLAYERVELOCITY
326.                 if keys[pg.K_s]:
327.                     self.vy = PLAYERVELOCITY
328.                 if keys[pg.K_d]:
329.                     self.vx = PLAYERVELOCITY
330.
331.                 if self.vx != 0 and self.vy != 0:
332.                     self.vx *= 0.7071
333.                     self.vy *= 0.7071
334.
335.             def rotate(self):
336.                 # The player rotates to face the mouse
337.                 self.mousex, self.mousey = pg.mouse.get_pos()
338.                 self.PLAYERCENTER = self.center
339.                 self.angle_vec = math.atan2((self.mousex - self.PLAYERCENTER[0]),(se
    lf.mousey - self.PLAYERCENTER[1]))
340.                 self.angle = math.degrees(self.angle_vec)
341.                 # Rotate the image
342.                 self.newimage = pg.transform.rotate(self.image, self.angle)
343.                 oldcenter = self.rect.center
344.                 self.newrect = self.newimage.get_rect()
345.                 self.newrect.center = oldcenter
346.
347.             def draw(self):
```

```
348.                 self.screen = maingame.returnGameScreen()
349.                 self.percentage = self.health / 100
350.                 xcoord = self.percentage * 416
351.                 # If the boss is not in that room draw the health bar at
352.                 # the bottom
353.                 if self.boss == None:
354.                     if self.health >= 0:
355.                         pg.draw.rect(self.screen,(0, 0, 0),[16, (HEIGHT -
    48), 416, 42])
356.                         pg.draw.rect(self.screen,(95, 99, 88),[20,(HEIGHT -
    48), 416, 38])
357.                         pg.draw.rect(self.screen,(227, 2, 43),[20,(HEIGHT -
    48), xcoord, 38])
358.                 # If in the bossroom the health bar is raised up
359.                 else:
360.                     if self.health >= 0:
361.                         pg.draw.rect(self.screen,(0, 0, 0),[70, (HEIGHT -
    112), 416, 42])
362.                         pg.draw.rect(self.screen,(95, 99, 88),[74,(HEIGHT -
    112), 416, 38])
363.                         pg.draw.rect(self.screen,(227, 2, 43),[74,(HEIGHT -
    112), xcoord, 38])
364.
365.                 self.screen.blit(self.newimage, self.newrect)
366.
367.
368.             def update(self, wall_group, closeddoor_group, closedEdoor_group, doors_
    group, Edoors_group, enemies, items, boss_projectiles,boss):
369.                 self.rect = self.hitbox_rect
370.                 self.boss = boss
371.                 self.keys()
372.                 self.rotate()
373.                 # Move player in the x coordinate
374.                 self.rect.x += self.vx
375.                 # - Collisions -
376.                 # Checks to see if the player can pick up a health pack
377.                 if self.health != 100:
378.                     # When collided +10 health is gained
379.                     if pg.sprite.spritecollide(self, items, True):
380.                         self.health += 10
381.                 # Enemy collision between player
382.                 collision =  pg.sprite.spritecollide(self, enemies, False)
383.
384.                 if collision:
385.                     # If the hit timer MOD 100 is 0 then take off 10 health
386.                     if (self.hittimer % 100) == 0 and self.hittimer != 0:
387.                         self.health = self.health - 10
388.                     # Takes off 10 health as soon as the hitboxes connect.
389.                     if self.hittimer == 0:
390.                         self.health = self.health - 10
391.
392.                     self.hittimer += 1
393.                 else:
394.                     self.hittimer = 0
395.                 # Checks to see if it's the boss room, by checking if the
396.                 # boss object is created
397.                 if boss != None:
398.                     boss_collision =  pg.sprite.spritecollide(self, boss, False)
399.
400.                     if boss_collision:
401.                         # If the hit timer MOD 100 is 0 then take
402.                         # off 25 health
403.                         if (self.boss_hittimer % 100) == 0 and self.boss_hittimer !=
     0:
404.                             self.health = self.health - 25
405.                         # Takes off 25 health as soon as the
```

```
406.                         # hitboxes connect.
407.                         if self.boss_hittimer == 0:
408.                             self.health = self.health - 25
409.
410.                         self.boss_hittimer += 1
411.                     else:
412.                         self.boss_hittimer = 0
413.                     # Checks to see if the projectiles of the boss hits the
414.                     # player and takes off 15 health
415.                     if pg.sprite.spritecollide(self, boss_projectiles, True):
416.                         self.health = self.health - 15
417.                 # Delete the object the player if the health falls below 0
418.                 if self.health <= 0:
419.                     self.kill()
420.                     return True
421.                 # Collisions for the walls in the map
422.                 self.collisionlist = pg.sprite.spritecollide(self, wall_group, False
    )
423.                 for collided in self.collisionlist:
424.                     # Checks to see is the velocity is greather than 0
425.                     # in the X plane
426.                     if self.vx > 0:
427.                         self.rect.right = collided.rect.left
428.                     else:
429.                         self.rect.left = collided.rect.right
430.                 # Colissions for the closed doors on the map
431.                 self.collisionlist2 = pg.sprite.spritecollide(self, closeddoor_group
    , False)
432.
433.                 for collided in self.collisionlist2:
434.                     if self.vx > 0:
435.                         self.rect.right = collided.rect.left
436.                     else:
437.                         self.rect.left = collided.rect.right
438.                 # Collision for the close Exit doors on the map
439.                 self.collisionlist3 = pg.sprite.spritecollide(self, closedEdoor_grou
    p, False)
440.                 for collided in self.collisionlist3:
441.                     if self.vx > 0:
442.                         self.rect.right = collided.rect.left
443.                     else:
444.                         self.rect.left = collided.rect.right
445.                 # Move player in the y coordinate
446.                 self.rect.y += self.vy
447.                 # Collisions for the walls in the map
448.                 self.collisionlist = pg.sprite.spritecollide(self, wall_group, False
    )
449.                 for collided in self.collisionlist:
450.                     # Checks to see is the velocity is greather than 0
451.                     # in the Y plane
452.                     if self.vy > 0:
453.                         self.rect.bottom = collided.rect.top
454.                     else:
455.                         self.rect.top = collided.rect.bottom
456.                 # Colissions for the closed doors on the map
457.                 self.collisionlist2 = pg.sprite.spritecollide(self, closeddoor_group
    , False)
458.                 for collided in self.collisionlist2:
459.                     if self.vy > 0:
460.                         self.rect.bottom = collided.rect.top
461.                     else:
462.                         self.rect.top = collided.rect.bottom
463.                 # Collision for the close Exit doors on the map for the "y" axis

464.                 self.collisionlist3 = pg.sprite.spritecollide(self, closedEdoor_grou
    p, False)
```

```
465.                 for collided in self.collisionlist3:
466.                     if self.vy > 0:
467.                         self.rect.bottom = collided.rect.top
468.                     else:
469.                         self.rect.top = collided.rect.bottom
470.
471.                 # Collision for the open doors on the map
472.                 self.doorcollisionlist = pg.sprite.spritecollide(self, doors_group,
    False)
473.                 if len(self.doorcollisionlist) > 0:
474.                     return 1
475.                 # Collision for the Exit open doors on the map, returns 2 when colli
    ded
476.                 self.doorlist = pg.sprite.spritecollide(self, Edoors_group, False)
477.                 if len(self.doorlist) > 0:
478.                     return 2
479.
480.
481.                 self.timer += 4
482.                 # This is where you a able to adjust the
483.                 # animation time, for the cycles of the pictures.
484.                 if (self.timer % 20) == 0:
485.
486.                     self.index += 1
487.                     if self.index >= len(self.imgindex):
488.                         self.index = 0
489.                     self.image = self.imgindex[self.index]
490.
491.             def LoadInto_OldMap(self, doordirection):
492.                 # check to see the door direction the load
493.                 # in to the coordinates
494.                 if doordirection == 1:
495.                     self.rect.x = (WIDTH - 144)
496.                     self.rect.y = (352)
497.                 elif doordirection == 2:
498.                     self.rect.x = (64)
499.                     self.rect.y = (352)
500.                 elif doordirection == 3:
501.                     self.rect.x = (480)
502.                     self.rect.y = (64)
503.                 elif doordirection == 4:
504.                     self.rect.x = (512 - 32)
505.                     self.rect.y = (HEIGHT - 144)
506.
507.             def LoadInto_NewMap(self, doordirection):
508.                 # check to see the door direction the load
509.                 # in to the coordinates
510.                 if doordirection == 1:
511.                     self.rect.x = (64)
512.                     self.rect.y = (352)
513.                 elif doordirection == 2:
514.                     self.rect.x = (WIDTH - 144)
515.                     self.rect.y = (352)
516.                 elif doordirection == 3:
517.                     self.rect.x = (512 - 32)
518.                     self.rect.y = (HEIGHT - 144)
519.                 elif doordirection == 4:
520.                     self.rect.x = (512 - 32)
521.                     self.rect.y = (64)
522.
523.             def getCenter(self):
524.                 # Return the player center
525.                 self.center = self.rect.center
526.                 return self.center
527.
528.         class MapGrid:
```

```
529.            def __init__(self, width, height):
530.                self.width = width
531.                self.height = height
532.                # Node connections in all four directions
533.                self.node_connections = [vector(1, 0), vector(-
    1, 0), vector(0, 1), vector(0, -1)]
534.                self.walls = []
535.                self.enemies = []
536.            def withinBoundary(self, node):
537.                # Check if node is within the screen boundary
538.                return 0 <= node.x < self.width and 0 <= node.y < self.height
539.            def passable(self, node):
540.                # Return true or false if the node is a wall
541.                return node not in self.walls
542.            def node_neighbours(self, node):
543.                neighbours = [node + connection for connection in self.node_connecti
    ons]
544.                if (node[0] + node[1]) % 2:
545.                    # Reverses objects in a list
546.                    neighbours.reverse()
547.                # Filters out nodes that are walls or outside the boundary
548.                neighbours = filter(self.withinBoundary, neighbours)
549.                neighbours = filter(self.passable, neighbours)
550.
551.                return neighbours
552.
553.        def aStar(graph, start, end):
554.            # Initialize a priority queue
555.            Pqueue = PriorityQueue()
556.            Pqueue.put((start), 0)
557.            # Initialzing path and cost dictionary
558.            path = {}
559.            cost = {}
560.            # Setting the starting node None and cost to 0
561.            path[(start)] = None
562.            cost[(start)] = 0
563.            # Iterate while the priority queue is not empty
564.            while not Pqueue.empty():
565.                current = Pqueue.get()
566.                if current == end:
567.                    # Break used to stop the astar search when the
568.                    # current node and goal node are the same
569.                    break
570.                # Check the next neighbouring nodes of the
571.                # current node
572.                for next in graph.node_neighbours(vector(current)):
573.                    next = vector_to_integer(next)
574.                    # Get the next node cost
575.                    next_cost = cost[current] + graph.cost(current, next)
576.                    if next not in cost or next_cost < cost[next]:
577.                        # Get the priority by adding the next cost and
578.                        # the hueristic value
579.                        priority = next_cost + heuristicValue(vector(end), vector(ne
    xt))
580.                        cost[next] = next_cost
581.                        # Puts the node into the priority queue
582.                        Pqueue.put(next, priority)
583.                        path[next] = vector(current) - vector(next)
584.            return path
585.
586.
587.
588.        def heuristicValue(a, b):
589.            # Return the hueristic value
590.            return (abs(a.x - b.x) + abs(a.y - b.y)) * 10
591.
```

```
        class WeightedGrid(MapGrid):
            def __init__(self, width, height):
                # Use of inheritance from the MapGrid
                super().__init__(width, height)
                self.weights = {}

            def cost(self, from_node, to_node):
                # Checks if the node distance is equal to 1, same as
                # all four directions of the node
                if (vector(to_node) - vector(from_node)).length_squared() == 1:
                    return self.weights.get(to_node, 0) + 10


        def breadthfirst_search(graph, start, end):
                # init queue
                queue = deque()
                queue.append(start)
                path = {}
                path[start] = None
                while len(queue) > 0:
                    current = queue.popleft()
                    if current == end:
                    # if the current node is at goal node
                        break
                    if graph.node_neighbours(current) != None:
                        for next in graph.node_neighbours(current):
                            # check if the node is not in path
                            if vector_to_integer(next) not in path:
                                queue.append(next)
                                path[vector_to_integer(next)] = current - next
                return path

        def vector_to_integer(v):
            # Returns an integer from a vector input, v
            return (int(v.x), int(v.y))

        class Room:
            def __init__(self, RoomNum, player, screen, direction, prevdirection):
                self.screen = screen
                self.RoomNum = RoomNum
                self.room = WeightedGrid(GRIDWIDTH, GRIDHEIGHT)
                self.walls = pg.sprite.Group()
                self.doors = pg.sprite.Group()
                self.ExitDoors = pg.sprite.Group()
                self.closedExitDoors = pg.sprite.Group()
                self.closeddoors = pg.sprite.Group()
                self.enemies = pg.sprite.Group()
                self.items = pg.sprite.Group()
                self.specialitems = pg.sprite.Group()
                self.player = player
                self.doordirection = direction
                # Gives restrictions on where objects
                # can spawn in specific rooms
                if RoomNum == 0:
                    self.roomLayout = [["B","B","B","B","B","B","B","B","B","B","B",
"B","B","B","B","B"],["B","","","","","","","","","","","","","","","B"],["B","",""
,"","","","","","","","","","","","B"],["B","","","","","","","","","","","","",
"","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","",
"","B","B","B","","","","","","B"],["B","","","","","","","B","P","B","","","","","
","B"],["B","","","","","","","B","B","B","","","","","","B"],["B","","","","","","
","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],
["B","","","","","","","","","","","","","","","B"],["B","B","B","B","B","B","B","B
","B","B","B","B","B","B","B","B"]]
                else:
```

```
            self.roomLayout = [["B","B","B","B","B","B","B","B","B","B","B","B","B","B","B","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","P","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","","","","","","","","","","","","","","","B"],["B","B","B","B","B","B","B","B","B","B","B","B","B","B","B","B"]]

        self.prevdoordirection = prevdirection
        self.iny = False
        self.inx = False

        self.door_replaced = False
        # Wall vectors on the outside boundary
        self.w = [(0,0), (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (9,0), (10,0), (11,0), (12,0), (13,0), (14,0), (15,0), (15,1), (15,2), (15,3), (15,4), (15,5), (15,6), (15,7), (15,8), (15,9), (15,10), (15,11), (14,11), (13,11), (12,11), (11,11), (10,11), (9,11), (8,11), (7,11), (6,11), (5,11), (4,11), (3,11), (2,11), (1,11), (0,11), (0,10), (0,9), (0,8), (0,7), (0,6), (0,5), (0,4), (0,3), (0,2), (0,1)]

        for wall in self.w:
            self.room.walls.append(vector(wall))

        if RoomNum != 5:
            self.AddItems()
            self.CreateRoomWalls()
            self.AddEnemies()
        self.CreateBoundary()
        #Checks for the last door direction so
        # it doesn't repeat and have two doors in one spot
        self.CheckPrevDirection()
        self.CheckCurrentDirection()

        pg.sprite.groupcollide(self.closeddoors, self.walls, False, True)
        pg.sprite.groupcollide(self.closedExitDoors, self.walls, False, True)

    def CheckCurrentDirection(self):
        # Assigning door image to the direction of the door
        if self.doordirection == 1:
            self.door = ClosedDoor(WIDTH - 64, 320, pg.image.load("closeddoorright.png").convert_alpha(), 1)
            self.closeddoors.add(self.door)

        elif self.doordirection == 2:
            self.door = ClosedDoor(32, 320, pg.image.load("closeddoorleft.png").convert_alpha(), 2)
            self.closeddoors.add(self.door)

        elif self.doordirection == 3:
            self.door = ClosedDoor(448, 48, pg.image.load("closeddoortop.png").convert_alpha(), 3)
            self.closeddoors.add(self.door)

        elif self.doordirection == 4:
            self.door = ClosedDoor(448, (HEIGHT - 64), pg.image.load("closeddoorbottom.png").convert_alpha(), 4)
            self.closeddoors.add(self.door)

        elif self.doordirection == 0:
            if self.prevdoordirection == 1:
                self.door = ClosedDoor(32, 320, pg.image.load("closeddoorleft.png").convert_alpha(), 2 )
```

```
                        self.closeddoors.add(self.door)
                    elif self.prevdoordirection == 2:
                        self.door = ClosedDoor(WIDTH - 64, 320, pg.image.load("closeddoorright.png").convert_alpha(), 1)
                        self.closeddoors.add(self.door)
                    elif self.prevdoordirection == 3:
                        self.door = ClosedDoor(448, (HEIGHT - 64), pg.image.load("closeddoorbottom.png").convert_alpha(), 4)
                        self.closeddoors.add(self.door)
                    elif self.prevdoordirection == 4:
                        self.door = ClosedDoor(448, 48, pg.image.load("closeddoortop.png").convert_alpha(), 3)
                        self.closeddoors.add(self.door)


            def CheckPrevDirection(self):
                # Checking for previous door direction
                if self.prevdoordirection == 1:
                    self.door = ClosedDoor(32, 320, pg.image.load("closeddoorleft.png").convert_alpha(), 2 )
                    self.closedExitDoors.add(self.door)
                elif self.prevdoordirection == 2:
                    self.door = ClosedDoor(WIDTH - 64, 320, pg.image.load("closeddoorright.png").convert_alpha(), 1)
                    self.closedExitDoors.add(self.door)
                elif self.prevdoordirection == 3:
                    self.door = ClosedDoor(448, (HEIGHT - 64), pg.image.load("closeddoorbottom.png").convert_alpha(), 4)
                    self.closedExitDoors.add(self.door)
                elif self.prevdoordirection == 4:
                    self.door = ClosedDoor(448, 48, pg.image.load("closeddoortop.png").convert_alpha(), 3)
                    self.closedExitDoors.add(self.door)

            def AddItems(self):
                # Add 0-2 items in a random spot in the room
                self.Item_Amount = random.randint(0, 2)
                self.itemnumber = 0
                if self.Item_Amount == 1:
                    # Probability of gettng the item
                    self.itemnumber = random.randint(1, 30)
                for i in range(self.Item_Amount):
                    validSpot = False
                    while validSpot == False:
                        item_ynumber = random.randint(1, 11)
                        item_xnumber = random.randint(1, 15)
                        if self.roomLayout[item_ynumber][item_xnumber] == "":
                            self.roomLayout[item_ynumber][item_xnumber] = "I"

                            if self.itemnumber >= 5:
                                self.sitem = Item((item_xnumber * TILESIZE),(item_ynumber * TILESIZE), 2)
                                self.specitem = (item_xnumber, item_ynumber)
                                self.specialitems.add(self.sitem)
                            else:
                                self.item = Item((item_xnumber * TILESIZE),(item_ynumber * TILESIZE), 1)
                                self.items.add(self.item)
                            validSpot = True

            def AddEnemies(self):
                self.enemylist = []
                # Add a random amount of enemies
                self.Enemy_Amount = random.randint(3 ,5)
                for i in range(self.Enemy_Amount):
                    validSpot = False
```

```
753.                     while validSpot == False:
754.                         # Generating a random x and y coordinate
755.                         ynumber = random.randint(3, 8)
756.                         xnumber = random.randint(3, 12)
757.                         if self.roomLayout[ynumber][xnumber] == "":
758.                             self.roomLayout[ynumber][xnumber] = "E"
759.                             # Initializing the enemy(s) into the game
760.                             self.enemy = Enemy(self.room, xnumber * TILESIZE,ynumber
    * TILESIZE)
761.                             # Adding enemies to enemy group
762.                             self.enemies.add(self.enemy)
763.                             self.enemylist.append(self.enemy)
764.
765.                             validSpot = True
766.                 # Minimum of 3 enemies
767.                 self.enemy1 = self.enemylist[0]
768.                 self.enemy2 = self.enemylist[1]
769.                 self.enemy3 = self.enemylist[2]
770.                 # If there are more than 3, enemies are added
771.                 # to the group respectively
772.
773.                 if self.Enemy_Amount > 3:
774.                     self.enemy4 = self.enemylist[3]
775.                 if self.Enemy_Amount > 4:
776.                     self.enemy5 = self.enemylist[4]
777.
778.
779.
780.             def CreateRoomWalls(self):
781.                 #Initilising the list layout of the map
782.                 validSpot = False
783.                 # the for loop gives the amount of objects
784.                 # you want in each room, max 6 otherwise an error occurs
785.                 # due to too many objects trying to fit in
786.                 for i in range(6):
787.                     validSpot = False
788.                     while validSpot == False:
789.                         # Gives a random chance to ge the
790.                         # different block types
791.                         blocktype = random.randint(1, 8)
792.                         ynumber = random.randint(3, 8)
793.                         xnumber = random.randint(3, 12)
794.                         if blocktype == 1:
795.
796.                             if self.roomLayout[ynumber][xnumber] == "":
797.                                 try:
798.                                     if  self.roomLayout[ynumber][xnumber + 2] == ""
    and self.roomLayout[ynumber + 1][xnumber + 2] == "" and self.roomLayout[ynumber ][x
    number - 1] == "" and self.roomLayout[ynumber + 1][xnumber -
    1  ] == "" and self.roomLayout[ynumber - 1 ][xnumber] == "" and self.roomLayout[ynu
    mber - 1 ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 2 ][xnumber] == "" and
     self.roomLayout[ynumber + 2 ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1
    ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1 ][xnumber] == ""  and self.ro
    omLayout[ynumber][xnumber + 1] == "":
799.                                         self.room.walls.append(vector(xnumber, ynumb
    er))
800.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber))
801.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber+1))
802.                                         self.room.walls.append(vector(xnumber, ynumb
    er+1))
803.                                         self.roomLayout[ynumber][xnumber] = "W"
804.                                         self.roomLayout[ynumber][xnumber + 1] = "W"
```

```
805.                                         self.roomLayout[ynumber + 1][xnumber] = "W"
806.                                         self.roomLayout[ynumber + 1 ][xnumber + 1] =
    "W"
807.                                         # 2x2 block
808.                                         self.wall = Wall(xnumber * TILESIZE,ynumber
    * TILESIZE, pg.image.load("normwall.png"))
809.                                         self.walls.add(self.wall)
810.                                         self.wall = Wall((xnumber+1) * TILESIZE,ynum
    ber * TILESIZE, pg.image.load("normwall.png"))
811.                                         self.walls.add(self.wall)
812.                                         self.wall = Wall((xnumber+1) * TILESIZE,(ynu
    mber+1) * TILESIZE, pg.image.load("normwall.png"))
813.                                         self.walls.add(self.wall)
814.                                         self.wall = Wall(xnumber * TILESIZE,(ynumber
    +1) * TILESIZE, pg.image.load("normwall.png"))
815.                                         self.walls.add(self.wall)
816.                                         validSpot = True
817.                                     except:
818.                                         print("error")
819.                                         validSpot = False
820.
821.                             if blocktype == 2:
822.
823.                                 if self.roomLayout[ynumber][xnumber] == "":
824.                                     try:
825.                                         if self.roomLayout[ynumber][xnumber + 2] == "" a
    nd self.roomLayout[ynumber + 1][xnumber + 2] == "" and self.roomLayout[ynumber ][xn
    umber - 1] == "" and self.roomLayout[ynumber + 1][xnumber -
    1  ] == "" and self.roomLayout[ynumber - 1 ][xnumber] == "" and self.roomLayout[ynu
    mber - 1 ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 2 ][xnumber] == "" and
     self.roomLayout[ynumber + 2 ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1
    ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1 ][xnumber] == ""  and self.ro
    omLayout[ynumber][xnumber + 1] == "":
826.                                             self.room.walls.append(vector(xnumber, ynumb
    er))
827.                                             self.room.walls.append(vector(xnumber+1, ynu
    mber))
828.                                             self.room.walls.append(vector(xnumber, ynumb
    er+1))
829.                                             self.roomLayout[ynumber][xnumber] = "W"
830.                                             self.roomLayout[ynumber][xnumber + 1] = "W"
831.                                             self.roomLayout[ynumber + 1][xnumber] = "W"
832.                                             # Right "L"
833.                                             self.wall = Wall(xnumber * TILESIZE,ynumber
    * TILESIZE, pg.image.load("normwall.png"))
834.                                             self.walls.add(self.wall)
835.                                             self.wall = Wall((xnumber+1) * TILESIZE,ynum
    ber * TILESIZE, pg.image.load("normwall.png"))
836.                                             self.walls.add(self.wall)
837.                                             self.wall = Wall((xnumber) * TILESIZE,(ynumb
    er+1) * TILESIZE, pg.image.load("normwall.png"))
838.                                             self.walls.add(self.wall)
839.                                             validSpot = True
840.                                     except:
841.                                         print("error")
842.                                         validSpot = False
843.                             if blocktype == 3:
844.
845.                                 if self.roomLayout[ynumber][xnumber] == "":
846.                                     try:
847.                                         if  self.roomLayout[ynumber][xnumber + 2] == ""
     and self.roomLayout[ynumber + 1][xnumber + 2] == "" and self.roomLayout[ynumber ][
    xnumber - 1] == "" and self.roomLayout[ynumber + 1][xnumber -
```

```
    1  ] == "" and self.roomLayout[ynumber - 1 ][xnumber] == "" and self.roomLayout[ynu
    mber - 1 ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 2 ][xnumber] == "" and
     self.roomLayout[ynumber + 2 ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1
    ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1 ][xnumber] == ""  and self.ro
    omLayout[ynumber][xnumber + 1] == "":
848.                                         self.room.walls.append(vector(xnumber, ynumb
    er))
849.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber))
850.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber+1))
851.
852.                                         self.roomLayout[ynumber][xnumber] = "W"
853.                                         self.roomLayout[ynumber][xnumber + 1] = "W"

854.                                         self.roomLayout[ynumber + 1][xnumber + 1] =
    "W"
855.                                         # Left "L"
856.                                         self.wall = Wall(xnumber * TILESIZE,ynumber
    * TILESIZE, pg.image.load("normwall.png"))
857.                                         self.walls.add(self.wall)
858.                                         self.wall = Wall((xnumber+1) * TILESIZE,ynum
    ber * TILESIZE, pg.image.load("normwall.png"))
859.                                         self.walls.add(self.wall)
860.                                         self.wall = Wall((xnumber + 1) * TILESIZE,(y
    number+1) * TILESIZE, pg.image.load("normwall.png"))
861.                                         self.walls.add(self.wall)
862.                                         validSpot = True
863.                                     except:
864.                                         print("error")
865.                                         validSpot = False
866.                             if blocktype == 4:
867.
868.                                 if self.roomLayout[ynumber][xnumber] == "":
869.                                     try:
870.                                         if self.roomLayout[ynumber][xnumber + 1] == "" a
    nd self.roomLayout[ynumber][xnumber - 1] == "" and self.roomLayout[ynumber + 1][xnu
    mber] == "" and self.roomLayout[ynumber - 1][xnumber] == "":
871.                                         # One Block
872.                                         self.room.walls.append(vector(xnumber, ynumb
    er))
873.                                         self.roomLayout[ynumber][xnumber] = "W"
874.                                         self.wall = Wall(xnumber * TILESIZE,ynumber
    * TILESIZE, pg.image.load("normwall.png"))
875.                                         self.walls.add(self.wall)
876.
877.                                         validSpot = True
878.                                     except:
879.                                         print("error")
880.                                         validSpot = False
881.
882.                             if blocktype == 5:
883.
884.                                 if self.roomLayout[ynumber][xnumber] == "":
885.                                     try:
886.                                          if  self.roomLayout[ynumber][xnumber + 2] == ""
     and self.roomLayout[ynumber + 1][xnumber + 1] == "" and self.roomLayout[ynumber ][
    xnumber - 1] == "" and self.roomLayout[ynumber - 1][xnumber -
    1  ] == "" and self.roomLayout[ynumber - 1 ][xnumber] == "" and self.roomLayout[ynu
    mber - 2 ][xnumber + 1 ] == "" and self.roomLayout[ynumber - 2 ][xnumber] == "" and
     self.roomLayout[ynumber - 1 ][xnumber + 2 ] == "" and self.roomLayout[ynumber - 1
    ][xnumber + 1 ] == "" and self.roomLayout[ynumber + 1 ][xnumber] == ""  and self.ro
    omLayout[ynumber][xnumber + 1] == "":
887.                                         self.room.walls.append(vector(xnumber, ynumb
    er))
```

```
888.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber -1))
889.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber))
890.
891.                                         self.roomLayout[ynumber][xnumber] = "W"
892.                                         self.roomLayout[ynumber][xnumber + 1] = "W"

893.                                         self.roomLayout[ynumber - 1][xnumber + 1] =
    "W"
894.                                         # right up "L"
895.                                         self.wall = Wall(xnumber * TILESIZE,ynumber
    * TILESIZE, pg.image.load("normwall.png"))
896.                                         self.walls.add(self.wall)
897.                                         self.wall = Wall((xnumber+1) * TILESIZE,ynum
    ber * TILESIZE, pg.image.load("normwall.png"))
898.                                         self.walls.add(self.wall)
899.                                         self.wall = Wall((xnumber + 1) * TILESIZE,(y
    number-1) * TILESIZE, pg.image.load("normwall.png"))
900.                                         self.walls.add(self.wall)
901.                                         validSpot = True
902.                                     except:
903.                                         print("error")
904.                                         validSpot = False
905.                             if blocktype == 6:
906.
907.                                 if self.roomLayout[ynumber][xnumber] == "":
908.                                     try:
909.                                          if  self.roomLayout[ynumber][xnumber + 2] == ""
     and self.roomLayout[ynumber + 1][xnumber + 1] == "" and self.roomLayout[ynumber ][
    xnumber - 1] == "" and self.roomLayout[ynumber - 1][xnumber -
    1  ] == "" and self.roomLayout[ynumber - 1 ][xnumber] == "" and self.roomLayout[ynu
    mber - 2 ][xnumber + 1 ] == "" and self.roomLayout[ynumber - 2 ][xnumber] == "" and
     self.roomLayout[ynumber -
    1 ][xnumber + 2 ] == "" and self.roomLayout[ynumber - 1 ][xnumber + 1 ] == "" and s
    elf.roomLayout[ynumber + 1 ][xnumber] == ""  and self.roomLayout[ynumber][xnumber +
     1] == "":
910.                                         self.room.walls.append(vector(xnumber, ynumb
    er))
911.                                         self.room.walls.append(vector(xnumber+1, ynu
    mber))
912.                                         self.room.walls.append(vector(xnumber, ynumb
    er -1))
913.                                         self.roomLayout[ynumber][xnumber] = "W"
914.                                         self.roomLayout[ynumber][xnumber + 1] = "W"

915.                                         self.roomLayout[ynumber - 1][xnumber] = "W"

916.                                         # left up "L"
917.                                         self.wall = Wall(xnumber * TILESIZE,ynumber
    * TILESIZE, pg.image.load("normwall.png"))
918.                                         self.walls.add(self.wall)
919.                                         self.wall = Wall((xnumber+1) * TILESIZE,ynum
    ber * TILESIZE, pg.image.load("normwall.png"))
920.                                         self.walls.add(self.wall)
921.                                         self.wall = Wall((xnumber) * TILESIZE,(ynumb
    er-1) * TILESIZE, pg.image.load("normwall.png"))
922.                                         self.walls.add(self.wall)
923.                                         validSpot = True
924.                                     except:
925.                                         print("error")
926.                                         validSpot = False
927.                             if blocktype == 7:
928.                                 if self.roomLayout[ynumber][xnumber] == "":
929.                                     try:
```

```
                                    if  self.roomLayout[ynumber][xnumber + 2] == "" and self.roomLayout[ynumber][xnumber + 1] == "" and self.roomLayout[ynumber][xnumber - 1] == "" and self.roomLayout[ynumber + 1][xnumber] == "" and self.roomLayout[ynumber - 1][xnumber] == "" and self.roomLayout[ynumber - 1][xnumber + 1] == "" and self.roomLayout[ynumber + 1][xnumber + 1] == "":
                                        self.room.walls.append(vector(xnumber, ynumber))
                                        self.room.walls.append(vector(xnumber+1, ynumber))

                                        self.roomLayout[ynumber][xnumber] = "W"
                                        self.roomLayout[ynumber][xnumber + 1] = "W"

                                        # Horizontal 2x1 wall
                                        self.wall = Wall(xnumber * TILESIZE,ynumber * TILESIZE, pg.image.load("normwall.png"))
                                        self.walls.add(self.wall)
                                        self.wall = Wall((xnumber+1) * TILESIZE,ynumber * TILESIZE, pg.image.load("normwall.png"))
                                        self.walls.add(self.wall)
                                        validSpot = True
                                    except:
                                        print("error")
                                        validSpot = False
                                if blocktype == 8:
                                    if self.roomLayout[ynumber][xnumber] == "":
                                        try:
                                            if  self.roomLayout[ynumber][xnumber + 1] == "" and self.roomLayout[ynumber][xnumber - 1] == "" and self.roomLayout[ynumber + 1][xnumber - 1] == "" and self.roomLayout[ynumber + 1][xnumber + 1] == "" and self.roomLayout[ynumber + 2][xnumber] == "" and self.roomLayout[ynumber - 1][xnumber] == "" and self.roomLayout[ynumber + 1][xnumber] == "":
                                                self.room.walls.append(vector(xnumber, ynumber))
                                                self.room.walls.append(vector(xnumber, ynumber +1))

                                                self.roomLayout[ynumber][xnumber] = "W"
                                                self.roomLayout[ynumber + 1][xnumber ] = "W"

                                                # Vertical block wall
                                                self.wall = Wall(xnumber * TILESIZE,ynumber * TILESIZE, pg.image.load("normwall.png"))
                                                self.walls.add(self.wall)
                                                self.wall = Wall((xnumber) * TILESIZE,(ynumber+1) * TILESIZE, pg.image.load("normwall.png"))
                                                self.walls.add(self.wall)
                                                validSpot = True
                                            except:
                                                print("error")
                                                validSpot = False
                    def CreateBoundary(self):
                        # Creates boundary of outer edge of the room
                        for i in range(16):
                            # Top boundary
                            self.wall = Wall(i * TILESIZE,0, pg.image.load("topwall.png"))
                            self.walls.add(self.wall)
                        for k in range(16):
                            # Bottom boundary
                            self.wall = Wall(k * TILESIZE,(HEIGHT - 64), pg.image.load("bottomwall.png"))
                            self.walls.add(self.wall)
                        for u in range(1, 11):
                            # Left boundary
                            self.wall = Wall(0, u * TILESIZE, pg.image.load("leftwall.png"))
```

```
976.                     self.walls.add(self.wall)
977.                 for o in range(1, 11):
978.                     # Right boundary
979.                     self.wall = Wall((WIDTH - 64), o * TILESIZE, pg.image.load("righ
    twall.png"))
980.                     self.walls.add(self.wall)
981.
982.             def update(self, proj):
983.                 # Checks to see if there is a special item in the room
984.                 if len(self.specialitems) > 0:
985.                     self.specialtrue = 1
986.                 else:
987.                     self.specialtrue = 0
988.                     self.specitem = None
989.                 doorcheck = 0
990.                 self.proj = proj
991.                 if self.RoomNum < 5:
992.                     # Update enemies in the room if the room is not number 5
993.                     self.enemy1.update(self.proj, self.player.rect.center, self.Play
    erNodeCheck(), self.walls, self.specialtrue, self.specitem)
994.                     self.enemynode1 = self.enemy1.get_node()
995.                     self.enemy2.update(self.proj, self.player.rect.center, self.Play
    erNodeCheck(), self.walls,self.specialtrue, self.specitem)
996.                     self.enemynode2 = self.enemy2.get_node()
997.                     self.enemy3.update(self.proj, self.player.rect.center, self.Play
    erNodeCheck(), self.walls, self.specialtrue, self.specitem)
998.                     self.enemynode3 = self.enemy3.get_node()
999.                     if self.Enemy_Amount > 3:
1000.                        self.enemy4.update(self.proj, self.player.rect.center, self.
    PlayerNodeCheck(), self.walls, self.specialtrue, self.specitem)
1001.                        self.enemynode4 = self.enemy4.get_node()
1002.                    if self.Enemy_Amount > 4:
1003.                        self.enemy5.update(self.proj, self.player.rect.center, self.
    PlayerNodeCheck(), self.walls, self.specialtrue, self.specitem)
1004.                        self.enemynode5 = self.enemy5.get_node()
1005.                    if self.Enemy_Amount > 5:
1006.                        self.enemy6.update(self.proj, self.player.rect.center, self.
    PlayerNodeCheck(), self.walls, self.specialtrue, self.specitem)
1007.                        self.enemynode6 = self.enemy6.get_node()
1008.                else:
1009.                    # Update the boss if it is room 5
1010.                    self.Boss.update(self.proj, self.walls)
1011.                # Allows the projectiles to have collisions between the room objects

1012.                pg.sprite.groupcollide(self.proj, self.walls, True, False)
1013.                pg.sprite.groupcollide(self.proj, self.closeddoors, True, False)
1014.                pg.sprite.groupcollide(self.proj, self.closedExitDoors, True, False)

1015.
1016.                if self.door_replaced == False:
1017.                    if len(self.enemies) == 0:
1018.                        if self.prevdoordirection == 1:
1019.                            self.door = Door(32, 320, pg.image.load("opendoorright.p
    ng").convert_alpha(), 2 )
1020.                            self.ExitDoors.add(self.door)
1021.                        elif self.prevdoordirection == 2:
1022.                            self.door = Door(WIDTH - 64, 320, pg.image.load("opendoo
    rright.png").convert_alpha(), 1)
1023.                            self.ExitDoors.add(self.door)
1024.                        elif self.prevdoordirection == 3:
1025.                            self.door = Door(448, (HEIGHT - 64), pg.image.load("open
    doortop.png").convert_alpha(), 4)
1026.                            self.ExitDoors.add(self.door)
1027.                        elif self.prevdoordirection == 4:
1028.                            self.door = Door(448, 32, pg.image.load("opendoortop.png
    ").convert_alpha(), 3)
```

```
1029.                             self.ExitDoors.add(self.door)
1030.                         ###############
1031.                         if self.doordirection == 1:
1032.                             self.door = Door(WIDTH - 64, 320, pg.image.load("opendoo
    rright.png").convert_alpha(), 1)
1033.                             self.doors.add(self.door)
1034.                         elif self.doordirection == 2:
1035.                             self.door = Door(32, 320, pg.image.load("opendoorright.p
    ng").convert_alpha(), 2)
1036.                             self.doors.add(self.door)
1037.                         elif self.doordirection == 3:
1038.                             self.door = Door(448, 32, pg.image.load("opendoortop.png
    ").convert_alpha(), 3)
1039.                             self.doors.add(self.door)
1040.                         elif self.doordirection == 4:
1041.                             self.door = Door(448, (HEIGHT - 64), pg.image.load("open
    doortop.png").convert_alpha(), 4)
1042.                             self.doors.add(self.door)
1043.                         pg.sprite.groupcollide(self.closeddoors, self.doors, True, F
    alse)
1044.                         pg.sprite.groupcollide(self.closedExitDoors, self.ExitDoors,
     True, False)
1045.                         self.door_replaced = True
1046.
1047.                 if self.RoomNum != 5:
1048.
1049.                     self.collidedwithdoor = self.player.update(self.walls, self.clos
    eddoors, self.closedExitDoors, self.doors, self.ExitDoors, self.enemies,self.items,
     None, None)
1050.                     if self.collidedwithdoor == 1:
1051.                         for projectile in self.proj:
1052.                             projectile.kill()
1053.                         self.player.LoadInto_NewMap(self.doordirection)
1054.                         return 1
1055.                     if self.collidedwithdoor == 2:
1056.                         for projectile in self.proj:
1057.                             projectile.kill()
1058.                         self.player.LoadInto_OldMap(self.prevdoordirection)
1059.                         return 2
1060.
1061.             def PlayerNodeCheck(self):
1062.                 # Need to be set to False again for when
1063.                 # the procedure is called again in the update section
1064.                 self.iny = False
1065.                 self.inx = False
1066.
1067.                 self.playercenter = self.player.getCenter()
1068.                 for i in range(0, WIDTH, TILESIZE):
1069.                     # Checking if the player's center "x"
1070.                     # coordinate is between a tile's area coordinates
1071.                     if i > self.playercenter[0] and (i - 64) < self.playercenter[0]
    :
1072.                         self.coordx = i
1073.                         self.inx = True
1074.
1075.                 for j in range(0, HEIGHT, TILESIZE):
1076.                     if j < self.playercenter[1] and (j + 64) > self.playercenter[1]
    :
1077.                         self.coordy = j
1078.                         self.iny = True
1079.
1080.                 # Searching through the tile list and
1081.                 # mapping out what tile the player's center is in
1082.                 if self.iny == True and self.inx == True:
1083.                     # Dividing the x and y coordinates
1084.                     # by 64 and minusing 1 to get into list form
```

```
                x = int(self.coordx / 64 - 1)
                y = int(self.coordy / 64)
                self.tempx = x
                self.tempy = y
                return (x, y)
            else:
                return (self.tempx, self.tempy)
        def draw(self):
            self.walls.draw(self.screen)
            self.specialitems.draw(self.screen)
            self.items.draw(self.screen)
            # Draw enemies when the room num is not 5
            if self.RoomNum < 5:
                self.enemy1.draw()
                self.enemy2.draw()
                self.enemy3.draw()
                if self.Enemy_Amount > 3:
                    self.enemy4.draw()
                if self.Enemy_Amount > 4:
                    self.enemy5.draw()
            self.closeddoors.draw(self.screen)
            self.closedExitDoors.draw(self.screen)
            if self.RoomNum != 5:

                if len(self.enemies) == 0:
                    self.doors.draw(self.screen)
                    self.ExitDoors.draw(self.screen)

    class Boss(pg.sprite.Sprite):
        def __init__(self, startposx, startposy, player, health = 1200):
            pg.sprite.Sprite.__init__(self)
            self.boss_projectiles = pg.sprite.Group()
            self.imgindex = []
            # Initalizing all images
            self.imgindex.append(pg.image.load("boss.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss2.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss3.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss4.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss5.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss6.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss7.v2.png").convert_alpha())
            self.imgindex.append(pg.image.load("boss8.v2.png").convert_alpha())
            self.index = 0
            self.image = self.imgindex[self.index]
            self.rect = self.image.get_rect()
            self.rect.x = startposx
            self.rect.y = startposy
            self.health = health
            self.player = player
            self.vx, self.vy = 0, 0
            self.first_movement = False
            self.direction = random.randint(1, 4)
            self.timer = 0
            self.stimer = 0
            self.wait = 0
            self.screen = maingame.returnGameScreen()
        def update(self, Group, wall_group):
            # Boss dies at 0 health
```

```
1143.             if self.health == 0:
1144.                 self.kill()
1145.                 return True
1146.             if pg.sprite.spritecollide(self, Group, True):
1147.                 self.health = self.health - 15
1148.             # Setting the hit timer for boss
1149.             if self.timer != 150:
1150.                 self.Boss_AttackCycle()
1151.                 self.rect.x += self.vx
1152.                 self.rect.y += self.vy
1153.                 # Directions to bounce
1154.                 if self.rect.x == 64 or self.rect.x == (WIDTH - 256):
1155.                     if self.direction == 1:
1156.                         self.direction = 4
1157.                     elif self.direction == 2:
1158.                         self.direction = 3
1159.                     elif self.direction == 3:
1160.                         self.direction = 2
1161.                     elif self.direction == 4:
1162.                         self.direction = 1
1163.                 if self.rect.y == 64 or self.rect.y == (HEIGHT - 256):
1164.                     if self.direction == 1:
1165.                         self.direction = 3
1166.                     elif self.direction == 2:
1167.                         self.direction = 4
1168.                     elif self.direction == 3:
1169.                         self.direction = 1
1170.                     elif self.direction == 4:
1171.                         self.direction = 2
1172.                 self.timer += 1
1173.             else:
1174.                 self.vx, self.vy = 0, 0
1175.                 if self.wait != 50:
1176.                     self.wait += 1
1177.                     if self.wait == 25:
1178.                         self.shoot_projectiles()
1179.                     elif self.wait == 49:
1180.                         self.shoot_projectiles()
1181.                 else:
1182.                     self.wait = 0
1183.                     self.timer = 0
1184.             self.playercenter = self.player.getCenter()
1185.             self.boss_projectiles.update()
1186.             self.rotatetowardsPlayer()
1187.             self.random_timer = random.randint(20, 25)
1188.             self.stimer += self.random_timer
1189.             # This is where you a able to adjust the
1190.             # animation time, for the cycles of the pictures.
1191.             if (self.stimer % 5) == 0:
1192.                 self.index += 1
1193.                 if self.index >= len(self.imgindex):
1194.                     self.index = 0
1195.                 self.image = self.imgindex[self.index]
1196.         def rotatetowardsPlayer(self):
1197.             self.angle_vec = math.atan2((self.rect.center[0] - self.playercenter
    [0]),(self.rect.center[1] - self.playercenter[1]))
1198.             # The angle is converted from radians to degrees
1199.             self.angle = math.degrees(self.angle_vec)
1200.             self.newimage = pg.transform.rotate(self.image, self.angle - 180)
1201.             oldcenter = self.rect.center
1202.             self.newrect = self.newimage.get_rect()
1203.             self.newrect.center = oldcenter
1204.
1205.         def shoot_projectiles(self):
1206.             self.boss_center = self.rect.center
```

```
1207.                 self.newproj = Projectile(self.boss_center[0], self.boss_center[1],
    self.playercenter[0] + 128, self.playercenter[1] + 128, 10, self.screen, 2)
1208.                 self.boss_projectiles.add(self.newproj)
1209.                 # Object added to a group
1210.                 self.newproj = Projectile(self.boss_center[0], self.boss_center[1],
    self.playercenter[0], self.playercenter[1], 10, self.screen, 2)
1211.                 self.boss_projectiles.add(self.newproj)
1212.                 # Object added to a group
1213.                 self.newproj = Projectile(self.boss_center[0], self.boss_center[1],
    self.playercenter[0] - 128, self.playercenter[1] - 128, 10, self.screen, 2)
1214.                 self.boss_projectiles.add(self.newproj)
1215.                 # Object added to a group
1216.
1217.             def Boss_AttackCycle(self):
1218.                 # Moving the boss along the direction
1219.                 if self.direction == 2:
1220.                     self.vx, self.vy = 4, 4
1221.                     self.vx *= -1
1222.                     self.vy *= -1
1223.                 if self.direction == 1:
1224.                     self.vx, self.vy = -4, -4
1225.                     self.vx *= -1
1226.                     self.vy *= -1
1227.                 if self.direction == 3:
1228.                     self.vx, self.vy = -4, 4
1229.                     self.vx *= -1
1230.                     self.vy *= -1
1231.                 if self.direction == 4:
1232.                     self.vx, self.vy = 4, -4
1233.                     self.vx *= -1
1234.                     self.vy *= -1
1235.
1236.             def draw(self):
1237.                 self.percentage = self.health / 1200
1238.                 xcoord = self.percentage * 512
1239.                 if self.health != 0:
1240.                     self.boss_projectiles.draw(self.screen)
1241.                     # draw health bar
1242.                     pg.draw.rect(self.screen,(0, 0, 0),[253, 13, 518, 38])
1243.                     pg.draw.rect(self.screen,(95, 99, 88),[256, 16, 512, 32])
1244.                     pg.draw.rect(self.screen,(227, 2, 43),[256, 16, xcoord, 32])
1245.                     try:
1246.                         self.screen.blit(self.newimage, self.newrect)
1247.                     except:
1248.                         self.screen.blit(self.image, self.rect)
1249.
1250.         # Creating an object which inherits from the class Room
1251.         class BossRoom(Room):
1252.             def __init__(self, RoomNum, player, screen, direction, prevdirection):
1253.                 # Super allows the parameters to be
1254.                 # taken from the Room class while allowing to
1255.                 # have its own __init__ to add differnt
1256.                 # more specific parameters.
1257.                 super().__init__(RoomNum, player, screen, direction, prevdirection)

1258.                 self.boss_group = pg.sprite.Group()
1259.                 self.AddBoss()
1260.                 self.player = player
1261.                 self.boss_dead = False
1262.                 self.winimage = pg.image.load("winscreen.png").convert_alpha()
1263.             def AddBoss(self):
1264.                 # Add the boss to the room
1265.                 self.Boss = Boss(448, 320, self.player)
1266.                 self.boss_group.add(self.Boss)
1267.
1268.             def update(self, projectile):
```

```
1269.               # Updating the bossroom and the projectile in it
1270.               super(BossRoom, self).update(projectile)
1271.               self.collidedwithdoor = self.player.update(self.walls, self.closeddo
    ors, self.closedExitDoors, self.doors, self.ExitDoors, self.enemies, self.items, se
    lf.Boss.boss_projectiles, self.boss_group)
1272.               if self.collidedwithdoor == 1:
1273.                   for projectile in self.proj:
1274.                       projectile.kill()
1275.                   self.player.LoadInto_NewMap(self.doordirection)
1276.                   return 1
1277.               if self.collidedwithdoor == 2:
1278.                   for projectile in self.proj:
1279.                       projectile.kill()
1280.                   self.player.LoadInto_OldMap(self.prevdoordirection)
1281.                   return 2
1282.               # Check to see if the boss is dead
1283.               if len(self.boss_group) == 0:
1284.                   self.boss_dead = True
1285.           def draw(self):
1286.               # Draw procedure for the boss room
1287.               super(BossRoom, self).draw()
1288.               self.screen = maingame.returnGameScreen()
1289.               if self.boss_dead == True:
1290.                   # Draw the win screen once boss is beaten
1291.                   self.screen.blit(self.winimage, (0,0))
1292.               else:
1293.                   self.walls.draw(self.screen)
1294.                   self.Boss.draw()
1295.
1296.               self.screen = maingame.returnGameScreen()
1297.
1298.       class Game:
1299.           def __init__(self):
1300.               # initializing the main object
1301.               pg.init()
1302.               self.screen = pg.display.set_mode((WIDTH, HEIGHT))
1303.               pg.display.set_caption("Dungeon Game")
1304.               self.clock = pg.time.Clock()
1305.               self.bg = pg.image.load("back.png").convert_alpha()
1306.               self.RoomNum = 0
1307.
1308.
1309.           def gameintro(self):
1310.               # load intro screen
1311.               self.image = pg.image.load("intropic.png")
1312.               timer = 0
1313.               intro_stage = True
1314.               while intro_stage == True:
1315.                   for event in pg.event.get():
1316.                       if event.type == pg.QUIT:
1317.                           self.quitGame()
1318.                       if event.type == pg.MOUSEBUTTONUP:
1319.                           intro_stage = False
1320.                   if timer < 250:
1321.                       self.screen.fill((255,255,255))
1322.                   if timer == 250:
1323.                       self.screen.fill((209, 209, 209))
1324.                   if timer == 500:
1325.                       self.screen.fill((161, 161, 161))
1326.                   if timer == 750:
1327.                       self.screen.fill((112, 112, 112))
1328.                   if timer >= 1000:
1329.                       self.screen.fill((89, 89, 89))
1330.                       self.screen.blit(self.image, (32,32))
1331.                   timer += 1
1332.                   pg.display.flip()
```

```
1333.          def deathscreen(self):
1334.              # load death screen
1335.              self.image = pg.image.load("deathscreen.png")
1336.              timer = 0
1337.              intro_stage = True
1338.              while intro_stage == True:
1339.                  for event in pg.event.get():
1340.                      if event.type == pg.QUIT:
1341.                          self.quitGame()
1342.                  if timer < 250:
1343.                      self.screen.fill((255,255,255))
1344.                  if timer == 250:
1345.                      self.screen.fill((209, 209, 209))
1346.                  if timer == 500:
1347.                      self.screen.fill((161, 161, 161))
1348.                  if timer == 750:
1349.                      self.screen.fill((112, 112, 112))
1350.                  if timer >= 1000:
1351.                      self.screen.fill((89, 89, 89))
1352.                      self.screen.blit(self.image, (32,32))
1353.                  timer += 1
1354.                  pg.display.flip()
1355.
1356.
1357.          def CheckforOppositeDoorDirection(self):
1358.              # checking what the opposite door direction is
1359.              if self.doordirection == 1:
1360.                  self.doordirection = random.randint(1, 4)
1361.                  if self.doordirection == 2:
1362.                      self.doordirection += 1
1363.              elif self.doordirection == 2:
1364.                  self.doordirection = random.randint(1, 4)
1365.                  if self.doordirection == 1:
1366.                      self.doordirection += 1
1367.              elif self.doordirection == 3:
1368.                  self.doordirection = random.randint(1, 4)
1369.                  if self.doordirection == 4:
1370.                      self.doordirection -= 2
1371.              elif self.doordirection == 4:
1372.                  self.doordirection = random.randint(1, 4)
1373.                  if self.doordirection == 3:
1374.                      self.doordirection -= 2
1375.
1376.          def CreateNewGame(self):
1377.              # Creating new sprite groups
1378.              self.projectiles = pg.sprite.Group()
1379.              self.player_group = pg.sprite.Group()
1380.              #Initilizing the player into the game
1381.              # with coordinates as parameters
1382.              self.player = PlayerSprite(512, 384)
1383.              self.player_group.add(self.player)
1384.              self.doordirection = random.randint(1, 4)
1385.
1386.              self.exitdoor2 = self.doordirection
1387.              # Initializing each room in the game
1388.              self.Room_0 = Room(self.RoomNum, self.player, self.screen, self.door
    direction, 0)
1389.              self.CheckforOppositeDoorDirection()
1390.              self.exitdoor3 = self.doordirection
1391.
1392.              self.Room_1 = Room(self.RoomNum + 1, self.player, self.screen,self.d
    oordirection, self.exitdoor2)
1393.              self.CheckforOppositeDoorDirection()
1394.              self.exitdoor4 = self.doordirection
1395.
```

```
1396.                 self.Room_2 = Room(self.RoomNum + 2, self.player, self.screen,self.d
    oordirection, self.exitdoor3)
1397.                 self.CheckforOppositeDoorDirection()
1398.                 self.exitdoor5 = self.doordirection
1399.
1400.                 self.Room_3 = Room(self.RoomNum + 3, self.player, self.screen,self.d
    oordirection, self.exitdoor4)
1401.                 self.CheckforOppositeDoorDirection()
1402.                 self.exitdoor6 = self.doordirection
1403.
1404.                 self.Room_4 = Room(self.RoomNum + 4, self.player, self.screen,self.d
    oordirection, self.exitdoor5)
1405.                 self.CheckforOppositeDoorDirection()
1406.
1407.                 self.Room_5 = BossRoom(self.RoomNum +5, self.player, self.screen,0,s
    elf.exitdoor6 )
1408.
1409.             def returnGameScreen(self):
1410.                 self.screen = self.screen
1411.                 return self.screen
1412.                 # Returns the game's screen
1413.             def drawBackground(self):
1414.                 self.screen.blit(self.bg, (0,0))
1415.
1416.             def MainGameLoop(self):
1417.                 self.gameRunning  = True
1418.                 # Main Game Loop
1419.                 while self.gameRunning:
1420.                     # Setting the clock tick rate to 60 ticks
1421.                     self.clock.tick(60)
1422.                     self.getEvents()
1423.                     self.update()
1424.                     self.CreateImage()
1425.
1426.
1427.             def CreateImage(self):
1428.                 # Drawing sub-section of the main loop
1429.                 self.drawBackground()
1430.                 self.projectiles.draw(self.screen)
1431.                 # Each room has a different object with
1432.                 # different data to keep the
1433.                 # data in that room's required 'data pack'
1434.                 if self.RoomNum == 0:
1435.                     self.Room_0.draw()
1436.                 elif self.RoomNum == 1:
1437.                     self.Room_1.draw()
1438.                 elif self.RoomNum == 2:
1439.                     self.Room_2.draw()
1440.                 elif self.RoomNum == 3:
1441.                     self.Room_3.draw()
1442.                 elif self.RoomNum == 4:
1443.                     self.Room_4.draw()
1444.                 elif self.RoomNum == 5:
1445.                     self.player.draw()
1446.                     self.Room_5.draw()
1447.                 if self.RoomNum != 5:
1448.                     self.player.draw()
1449.
1450.                 # Flips the display at the end to change the image
1451.                 pg.display.flip()
1452.
1453.             def AimLine(self):
1454.                 # Testing attribute to visually see the vector
1455.                 # of the player's aim
1456.                 pg.draw.line(self.screen, (0, 0, 0), (self.mousex, self.mousey), (se
    lf.PLAYERCENTER))
```

```
1457.
1458.          def DrawGrid(self):
1459.              # Draws a grid with gives a reference for testing
1460.              for i in range(0, WIDTH, TILESIZE):
1461.                  pg.draw.line(self.screen, (0, 0, 0), (i, 0), (i, HEIGHT))
1462.              for j in range(0, HEIGHT, TILESIZE):
1463.                  pg.draw.line(self.screen, (0, 0, 0), (0, j), (WIDTH, j))
1464.          def getEvents(self):
1465.              self.PLAYERCENTER = self.player.getCenter()
1466.              self.mousex, self.mousey = pg.mouse.get_pos()
1467.              for event in pg.event.get():
1468.                  self.mouse = pg.mouse.get_pressed()
1469.                  if event.type == pg.MOUSEBUTTONUP:
1470.                      if event.button == 1:
1471.                          # Doesn't allow more than 5 projectiles on
1472.                          # screen at once
1473.                          if len(self.projectiles) < 5:
1474.                              # Creates new projectile object
1475.                              self.newproj = Projectile(self.PLAYERCENTER[0], self
    .PLAYERCENTER[1], self.mousex, self.mousey, 15, self.screen, 1)
1476.                              self.projectiles.add(self.newproj)
1477.                              # Object added to a group
1478.                  # When the top right cross is clicked, the
1479.                  # program closes
1480.                  if event.type == pg.QUIT:
1481.                      self.quitGame()
1482.          def update(self):
1483.              # Check to see if the player is dead
1484.              if self.player.health <= 0:
1485.                  self.gameRunning  = False
1486.              self.projectiles.update()
1487.              # Update section for all rooms
1488.              if self.RoomNum == 0:
1489.                  if self.Room_0.update(self.projectiles) == 1:
1490.                      self.RoomNum = 1
1491.              elif self.RoomNum == 1:
1492.                  door1 = self.Room_1.update(self.projectiles)
1493.                  if door1 == 1:
1494.                      self.RoomNum = 2
1495.                  if door1 == 2:
1496.                      self.RoomNum -= 1
1497.              elif self.RoomNum == 2:
1498.                  door2 = self.Room_2.update(self.projectiles)
1499.                  if door2 == 1:
1500.                      self.RoomNum = 3
1501.                  if door2 == 2:
1502.                      self.RoomNum -= 1
1503.              elif self.RoomNum == 3:
1504.                  door3 = self.Room_3.update(self.projectiles)
1505.                  if door3 == 1:
1506.                      self.RoomNum = 4
1507.                  if door3 == 2:
1508.                      self.RoomNum -= 1
1509.              elif self.RoomNum == 4:
1510.                  door4 = self.Room_4.update(self.projectiles)
1511.                  if door4 == 1:
1512.                      self.RoomNum = 5
1513.                  if door4 == 2:
1514.                      self.RoomNum -= 1
1515.              elif self.RoomNum == 5:
1516.                  if self.Room_5.update(self.projectiles) == 2:
1517.                      self.RoomNum = 4
1518.          # quit procedure
1519.          def quitGame(self):
1520.              pg.quit()
1521.              sys.exit()
```

```
1522.
1523.
1524.
1525.
1526.      # Creates the main game object
1527.      maingame = Game()
1528.      while True:
1529.          maingame.gameintro()
1530.          maingame.CreateNewGame()
1531.          maingame.MainGameLoop()
1532.          maingame.deathscreen()
1533.
1534.
```

System Maintenance

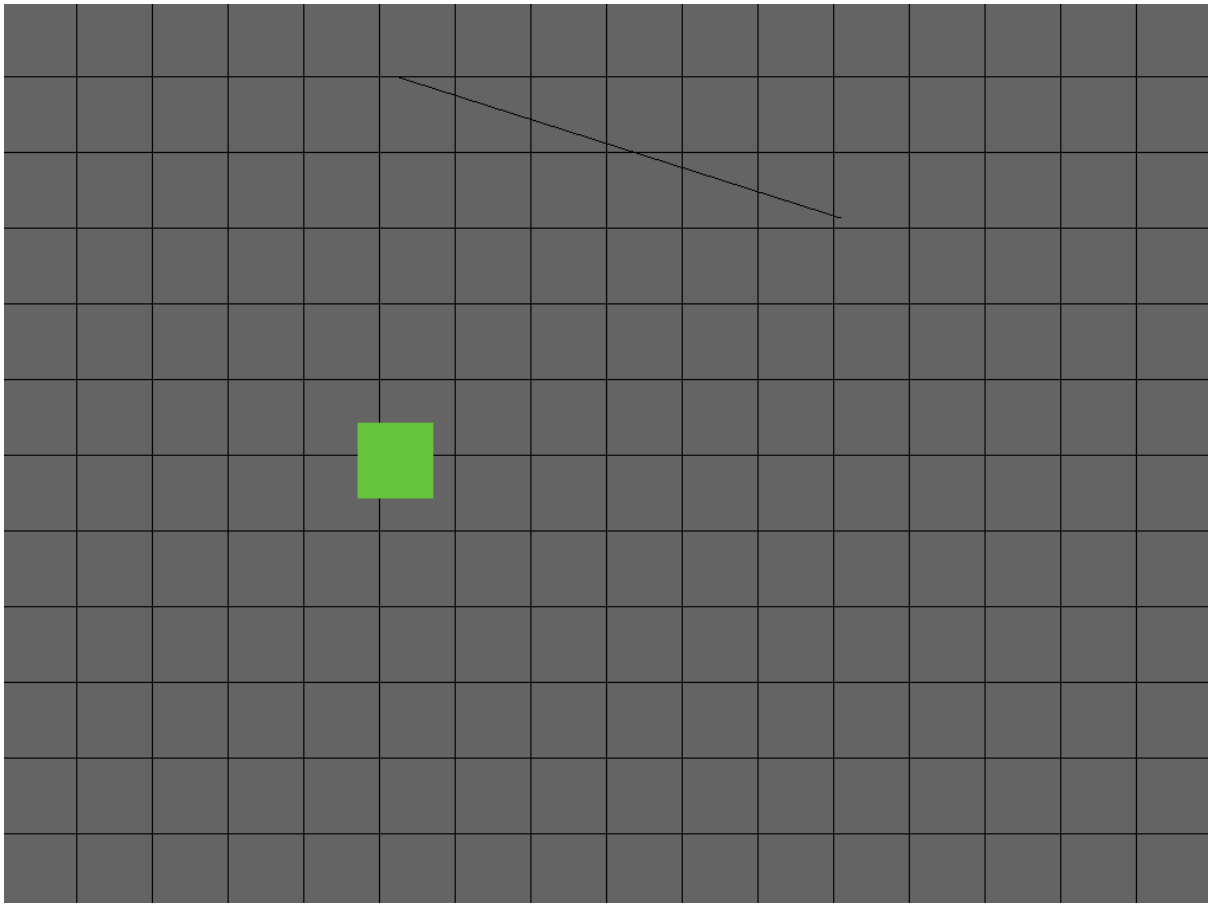
#1

To try and get the aiming line to work, it needed to have an updated position on the mouse cursor's X & Y coordinates and updated coordinates of the center of the sprite's rect. Despite this, I ran into a problem with the aim line not updating when a button was held down with the mouse not being moved.

When Holding down the button S the line would not follow the sprite, this was due to the sprite's center coordinates not being updated, which the code still thought that the coordinates where still in that position.

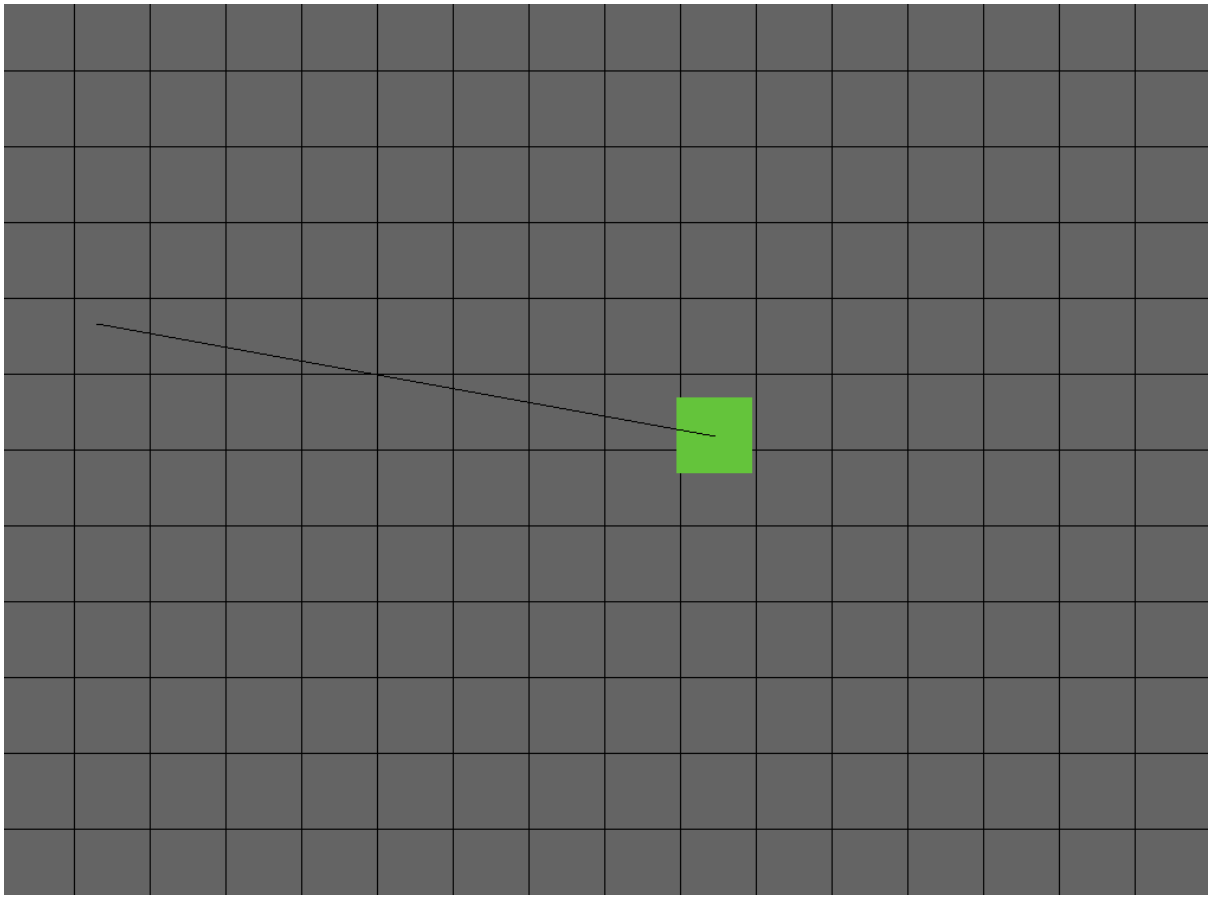
The source of the problem was that the game needed to update the sprite's movements before the game's main event line, so when holding a key down, the code wouldn't have a chance to go back to the main event line.

```
def getEvents(self):
    for event in pg.event.get():
        self.mousex, self.mousey = pg.mouse.get_pos()
        self.PLAYERCENTER = self.player.getCenter()
        if event.type == pg.QUIT:
            self.quitGame()
```



To fix this bug I made sure that the players center was updated before after the events took place, so the center was able to update with the sprite movement, even when holding the button down.

```
def update(self):
    self.PLAYERCENTER = self.player.getCenter()
    self.player.update()
```



#2

When shooting projectiles, the projectile needed to follow a straight path; in which a vector created between the mouse cursor and the player's center. The projectile, in theory, also need to head towards infinity, once the vector coordinates were paved.

Despite this, I coded this before I started working on collisions. So, the projectiles would move out of the screen's coordinates and carry on going. There were two problems with this:

1. Having many sprites that are actually there will make the performance of the game slow, due to the number of projectile sprites the program will have to constantly update then draw/render.
2. The projectiles would come back on screen eventually. I'm not 100% sure but I think that the pygame's surface is mapped out to be sphere like, so the projectiles would just come back to the original position.

To fix this, I made a sprite group for all the projectiles to go in and made a border check subroutine to make sure that the rect(hitbox) of the projectile was on the screen's dimensions, the check would check every time the projectile updated. If the projectile did go off the screen, the subroutine would return true and use the command self.kill(), this will eliminate that certain projectile so the problems will not be able to occur.

Check Subroutine

```
def OffScreencheck(self):
    if (self.rect[0] + 16) < 0 or self.rect[0] > WIDTH or (self.rect[1] + 16) < 0 or self.rect[1] > HEIGHT:
        return True
    else:
        return False
```

All this subroutine does is check the pixels of the projectile's hitbox and see if any arguments are met within all four directions.

```
<Group(0 sprites)>
Killed
<Group(0 sprites)>
Killed
<Group(0 sprites)>
Killed
```

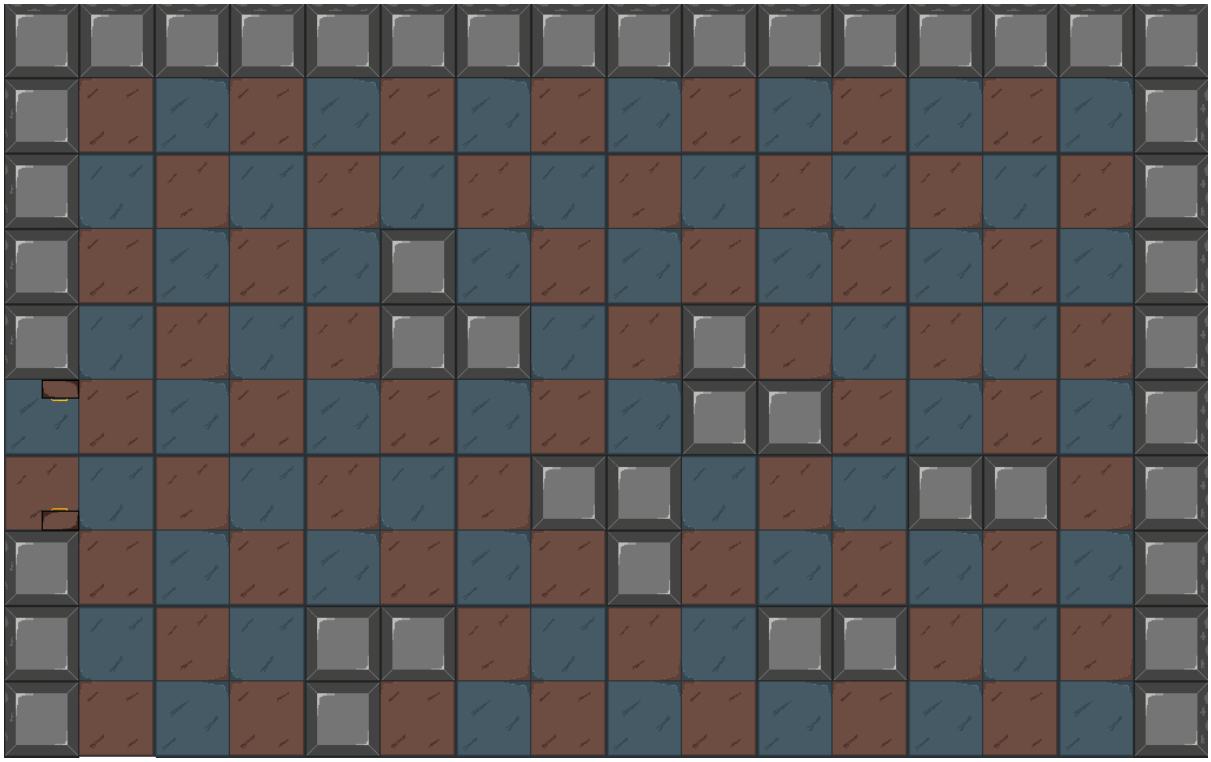
As seen above, I shot 3 separate projectiles and I put a simple print command in place as a check, but also printed out the number of sprites in the projectile group. And it shows that the sprites get deleted once they hit the border of the screen.

As I proceed on with the code, collisions with do most of this when two objects collide, but for now the games performance will not be hindered if there is a collision bug later on.

#3

When programming the collisions, I firstly settled to get the current coordinates and subtract the coordinates number values so when the hitboxes come into contact; the player's hitbox is moved so a certain amount of pixels so they don't come in contact. Despite this, I ran into a problem which gave the impression that the player was bouncing off the wall when colliding with another hitbox. From an aesthetic point of view the "bouncing" didn't look good, but there was another problem that raised a bug within the game.

If a hitbox was collided with another a certain way, the player would end up teleporting outside the boundaries of the map, therefore breaking the game.



To end up fixing this problem I used the pygame's built in sprite collide function to create a list of all collided sprites and check both x and y axis velocities and see if they're greater or less than zero, once determined, you would set the respective side, once collided, to the opposite side of the player's hitbox and have it so the .rect of that hitbox is the opposite of the other .rect in all four directions.

Now this is done, the hitboxes can glide smoothly other each other with no problems and no random "bouncing" movement occours.

#4

When coding I ran into a problem of the room size not being big enough for a certain number of walls. Despite this, I speculated that this would become a problem, so I found out that the maximum number of walls was 6, this was due to the biggest block(2x2) only having enough room for 6 of them will all the set rules that in place in the algorithm. You could fit more blocks on the map if they were just 1x1, but due to the random nature of the procedual block types this couldn't be accomplished.
When using 6+ block types a runtime error occurs where the validspot boolean check just keeps repeating under the while loop due to the program never finding a valid spot.

#5

When setting up the collisions between the sprites and the player I ran into a problem of the sprite instantly killing the player and ending the game. This was due to the collision list instantly taking the health off the player when they do collide and making it zero, ending the game.

```
 if collision:

         self.health = self.health - 10
```

To fix this I created a timer to only take away a certain amount of health in each time interval.

```
if collision:
    if (self.hittimer % 100) == 0 and self.hittimer != 0:
        self.health = self.health - 10

    if self.hittimer == 0:
        self.health = self.health - 10

    self.hittimer += 1
else:
    self.hittimer = 0
```

This when a collision happens, the code checks to see if the hit timer is in the 100s and it also checks to see if the hit timer does not equal zero. Every 100 interval of time, the player will lose 10 health points. Despite this, the secondary if statement will be the initial collision so it will take 10 health points one the first collision. With this, the hit timer will increase by one each time. Lastly, if no sprite is colliding with the with the player sprite the hit timer will be reset back to zero.

Testing

Test Strategy

My game will need to be tested thoroughly to make sure that there are no errors/bugs when the final product is finished. The testing must occur from a user's point of view but also a programming point of view, most of the testing will be proven by screen shot proof.

I will use three types of testing; Normal, Erroneous and Boundary. There will not be a lot of erroneous testing due to the user not being able to input a lot. The boundary testing will be focused on the number-based systems within the program. The normal testing will be focused on valid data to see if the game should work how it is intended to.

All the main aspects of the game will be tested. The player sprite will be thoroughly tested to make sure that the user experience is good and experiences no bugs. The enemies of the game will also be tested to see how they behave with pathfinding capabilities and with interactions with the room around them. Collisions between everything will also be tested thoroughly to make sure that nothing bugs out and no game-breaking glitches occur.

With the tests that take a branch of code, the proof will be a test print statement to show that the path of the code is taken and executed. This gives an insight of what the code is doing and path it's going to take.

If a test returns an incorrect actual outcome, I will look at the code and see what has went wrong and fix it. Once the code has been fixed, I will run the test again through the whole process and make sure that the expected outcome is the same as the actual outcome.

When showing proof of test, pictures will have stuff drawn on it to show you what is going to happen then a secondary screen shot of after the event has happened. Despite this, each test should be used to benefit the user's experience and overall make it better, so the program can run with minimal bugs and be very polished in the final product.

Normal Tests

Test No.	Purpose of the test	Test Data	Expected Outcome	Actual Outcome	Changes Needed	Screen shot Proof
1.1	To see if a new game creates when a mouse button is clicked	Click anywhere on the screen	The game should initialize and place the player right into the game straight away	The game does initialize/transition to the game straight away	N/A	Fig 1.1

1.2	To see if the “x” in the upper right exits out of the game and quits pygame	Click the “X” in the top right of the window	The window should close	The window closes and exits pygame	N/A	Fig 1.2
1.3	To see if you can still move around once you've beat the game with the “you win” screen	Beat the final boss in the final room	The player should still be able to move around with an overlay with “you win” drawn on top of the player	The player can move around under the “you win” overlay	N/A	Fig 1.3
1.4	To see if the “you lose” screen comes up	Get hit by enemies until the player sprite is at 0 health	The “you lose” screen should pop up.	The “you lose” screen does pop up	N/A	Fig 1.4
2.1.1	To see what direction you move when the key “w” is pressed	Press/hold the “w” key	The player sprite should move upwards	The player sprite moves upwards	N/A	Fig 2.1.1
2.1.2	To see what direction you move when the key “s” is pressed	Press/hold the “s” key	The player sprite should move downwards	The player sprite moves downwards	N/A	Fig 2.1.2
2.1.3	To see what direction you move when the key “a” is pressed	Press/hold the “a” key	The player sprite should move left	The player sprite moves left	N/A	Fig 2.1.3
2.1.4	To see what direction you move when the key “d” is pressed	Press/hold the “d” key	The player sprite should move right	The player sprite moves right	N/A	Fig 2.1.4
2.2.1	To see what direction you move when the key “w” and the key “a” is pressed at the same time	Press/hold the “w” and “a” key	The player sprite should move diagonally to the top left	The player sprite moves diagonally to the top left	N/A	Fig 2.2.1
2.2.2	To see what direction you move when the key “w” and	Press/hold the “w” and “d” key	The player sprite should move diagonally to the top right	The player sprite moves diagonally to the top right	N/A	Fig 2.2.2

	the key “d” is pressed at the same time					
2.2.3	To see what direction you move when the key “s” and the key “a” is pressed at the same time	Press/hold the “s” and “a” key	The player sprite should move diagonally to the bottom left	The player sprite moves diagonally to the bottom left	N/A	Fig 2.2.3
2.2.4	To see what direction you move when the key “w” and the key “d” is pressed at the same time	Press/hold the “s” and “d” key	The player sprite should move diagonally to the bottom right	The player sprite moves diagonally to the bottom right	N/A	Fig 2.2.4
2.3	To see if the player sprite image rotates towards the mouse	Move mouse around the player sprite	The sprite’s image will rotate towards the mouse cursor	The player rotates towards the mouse cursor correctly	N/A	Fig 2.3
2.4	To see if a projectile shoot and travels in the correct direction	Click the left mouse button within the window	The projectile will travel towards the mouse cursor	The projectile successfully travelled towards the mouse click	N/A	Fig 2.4
2.5	To see if the doors on the map open after all the enemies have been killed	Get rid of all enemies on the map.	The door(s) on the map will open to either go to the next or previous rooms	The doors open when all the enemies are gone	N/A	Fig 2.5
3.1	To see if the collision between the wall and the player work	Collide with the walls in all four directions to see if the player can pass through them	The player sprite will not be able to pass through the wall	The player sprite is unable to pass through the walls	N/A	Fig 3.1
3.2	To see if the collisions work between the player and non-boss enemies	Collide with an enemy	The health bar should lower, and 10 health should be taken off per hit	The player’s health decreased by 10 and the health bar decreased also	N/A	Fig 3.2
3.3	To see if the collision between the	Collide with open door sprite	The player should be put into a new room with enemies	The player is put into a new room with enemies.	N/A	Fig 3.3

	door and player works					
3.4	To see if the collisions between the player and health packs work	Collide with a health pack	If the player is less than 100 the player should collide with the health pack and gain 10 health with the health pack disappearing	The player does gain 10 health and the health pack does disappear	N/A	Fig 3.4
3.5	To see if the collisions between projectiles and enemies work	Shoot an enemy	The projectile should disappear, and the enemies' health should decrease by 10	The projectile does disappear, and the enemy's health does decrease by 10	N/A	Fig 3.5
3.6	To see if collisions between projectiles and walls work	Shoot a projectile towards a wall	The projectile should just disappear and nothing else should happen	The projectile does disappear and nothing else happens	N/A	Fig 3.6
3.7	To see if the projectile destroys itself when shot through and open door	Shoot a projectile through an open door	The projectile should travel through the door and destroy when the edge of the screen is reached	The projectile destroys when it goes off screen	N/A	Fig 3.7
3.8	To see if collisions work between the boss	Collide with the boss sprite	The player should have 25 taken off	The player's health is minuses by 25	N/A	Fig 3.8
4.1	To see if enemies use the correct type of path finding when there is a beacon item in the room.	Initialize a room with a beacon item in	The enemies should use breadth first search to path find towards the beacon. And the print check will print when that style of pathfinding is chosen by the enemy	The breadth first search is used, and the enemies path find to the beacon item.	N/A	Fig 4.1
4.2	To see if enemies use the correct type of pathfinding when there are no items within the room	Initialize a room without a beacon item In the room	The enemies should use a star search to path find towards the player. And the print check will print when that style of pathfinding is	The A star search is used, and the enemies path find towards the player.	N/A	Fig 4.2

			chosen by the enemy.			
4.3	To see if the enemies stop at the goal node	Don't move with the player	The enemies should crowd onto the goal node of the player.	The enemies all stick on the goal node unless it changes	N/A	4.3
4.4	To see if the enemies stop on the player during their path towards a beacon item	Move in front of the path of an enemy when a beacon is initialized in the room	The enemy should not proceed past the player and should stay at the goal node. If the player moves, the enemy must start pathfinding again towards the beacon	The enemy stays at the goal node and does not pass through the player. When the player moves the enemy proceeds to path find back towards the beacon node.	N/A	4.4

Erroneous Testing

Test No.	Purpose of the test	Test Data	Expected Outcome	Actual Outcome	Changes Needed	Screen shot Proof
E1.1	To test projectiles shot at 0,0	Shot a projectile when the mouse is exactly in the middle of the player sprite	The projectile should destroy	The projectile doesn't destroy and a ZeroDivisionError occurs	Need to put an except statement to destroy this object with this happens	Fig E1.1
E1.1 (corrected)	To test projectiles shot at 0,0	Shot a projectile when the mouse is exactly in the middle of the player sprite	The projectile should destroy	The projectile destroys	N/A	Fig E1.1 (corrected)
E2.1	To test putting more than 6 block types in the game	Put more than 6 block types into the procedural generation	The program will be stuck in a loop due to it not being able to find a valid spot	The program gets stuck and a loop.	Keep the program to a maximum of 6 block types.	Fig E2.1
E3.1	Pressing any of the other non-game buttons on the keyboard	Test random buttons on the keyboard/mouse	The buttons on the keyboard/mouse do not affect the game	The buttons on the keyboard/mouse do not affect the game	N/A	N/A

Boundary Testing

Test No.	Purpose of the	Test Data	Expected Outcome	Actual Outcome	Changes Needed	Screen shot Proof

	test					
B1.1	To test if you can pick up a health pack when you are already at 100 health(max)	Collide with a health pack when the player is at 100 health	The health pack should stay there, and the player should not gain any more health.	The player does not gain any extra health and the health pack does not disappear	N/A	Fig B1.1
B2.1	To test how many projectiles the player can shoot at once	Click recursively and see how many projectile objects can exist at once	Maximum of 5 projectile objects can only exist at once	5 projectile objects only exist at once.	N/A	Fig B2.1

Normal Test Screenshots

Fig 1.1

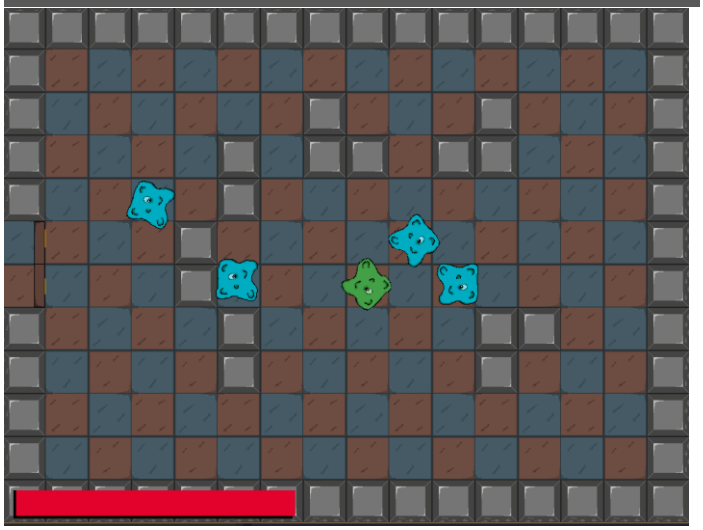


Fig 1.2

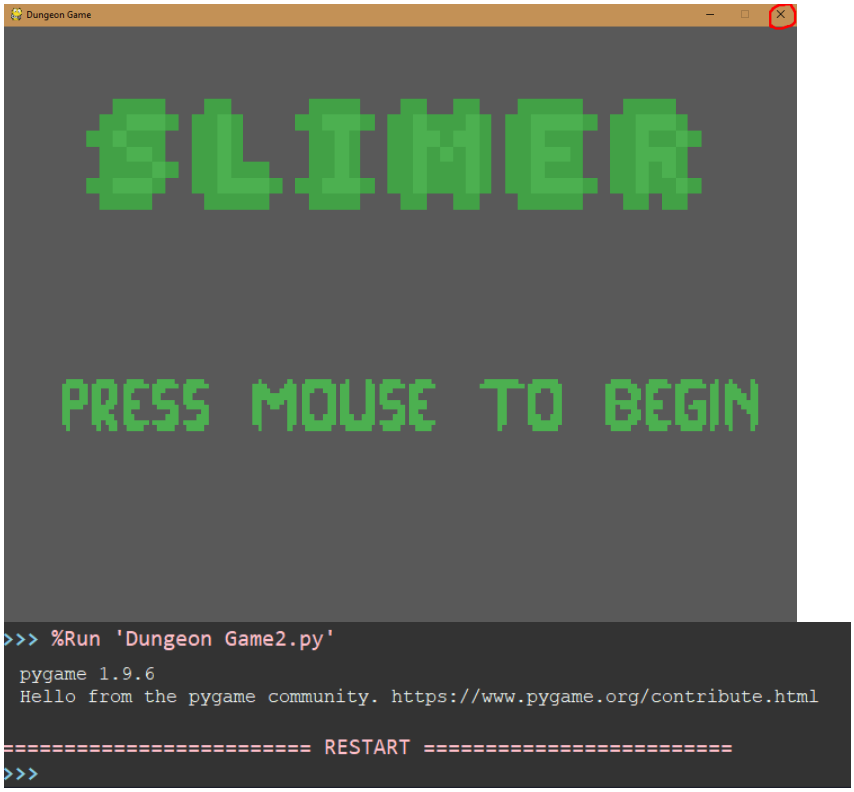


Fig 1.3

The two screenshots show the player moved to two different spots In the same instance.

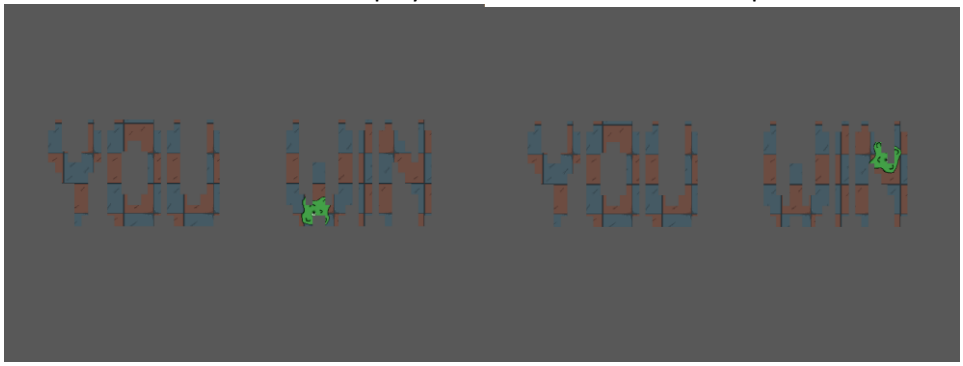


Fig 1.4

When the player sprite hits the enemies its health should go down to zero and that should toggle the "you lose" screen. The value on the right is the health printed.

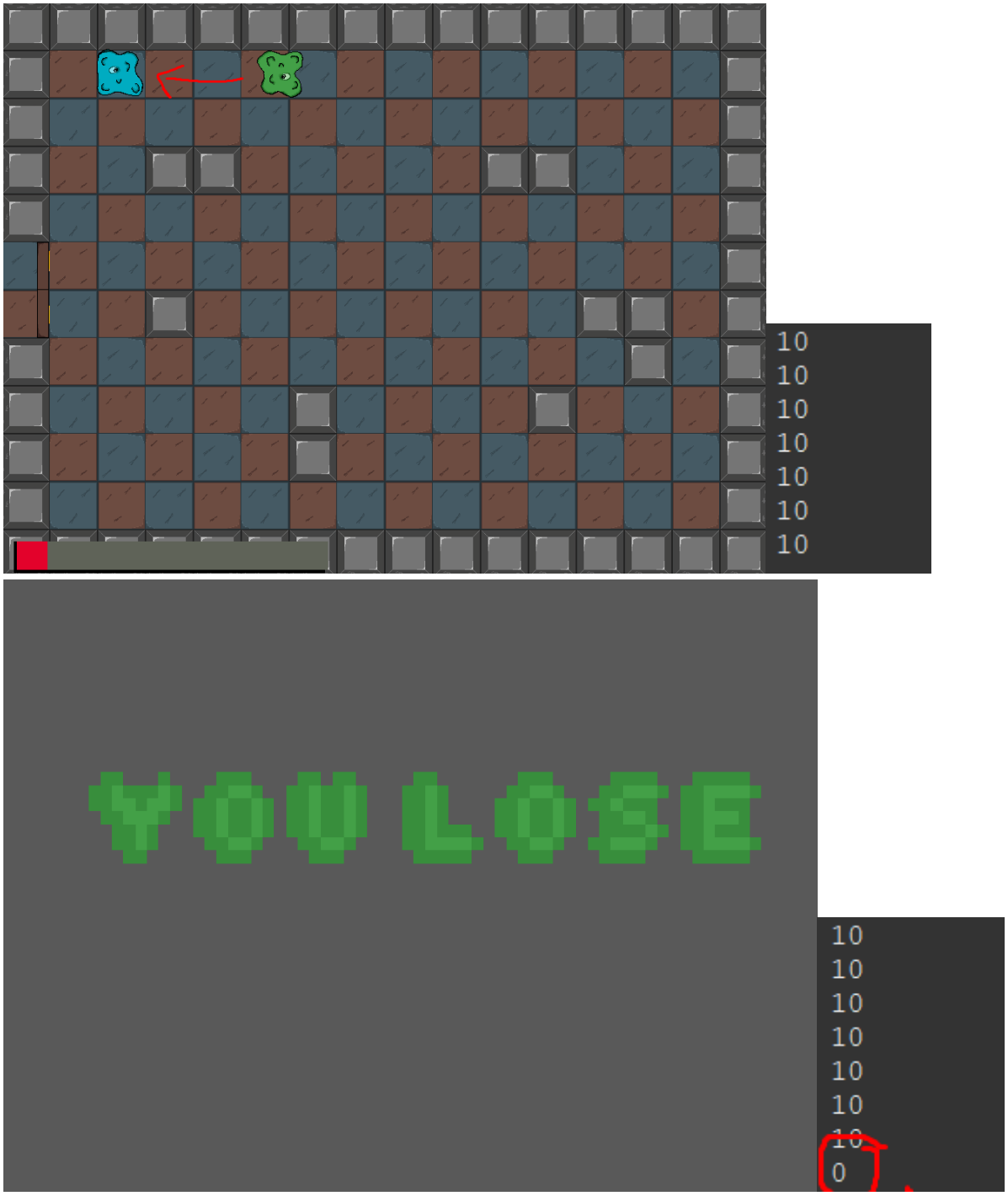


Fig 2.1.1
The player sprite moved upwards

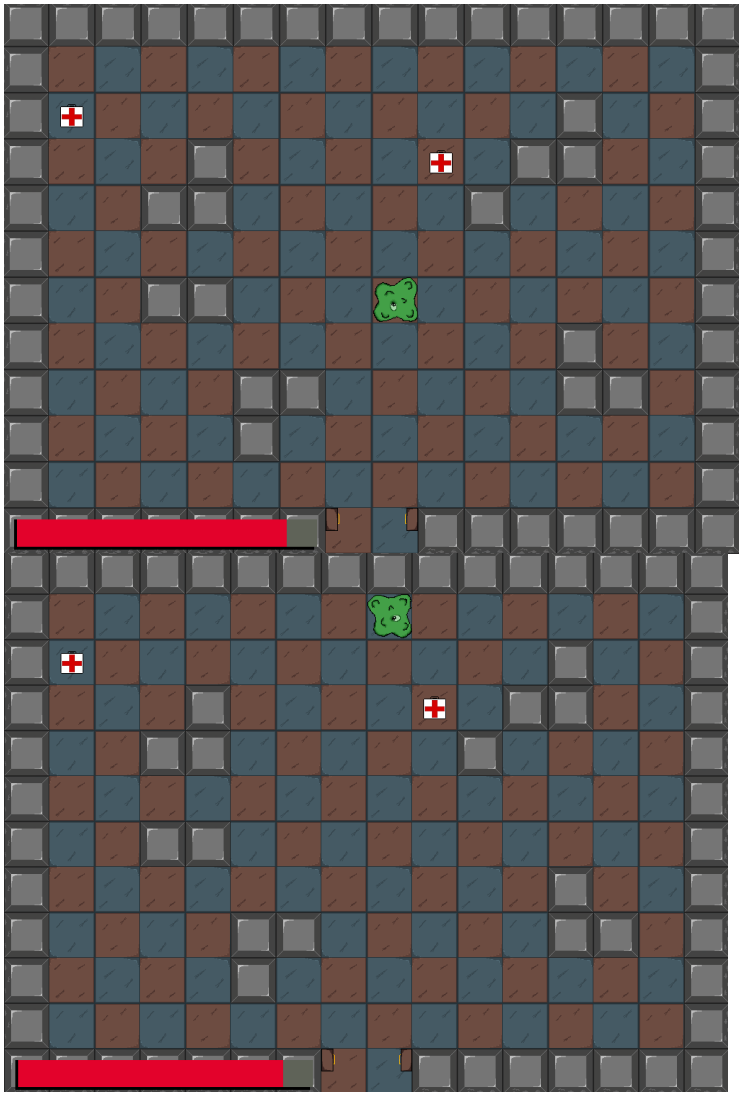


Fig 2.1.2
The player sprite moved downwards

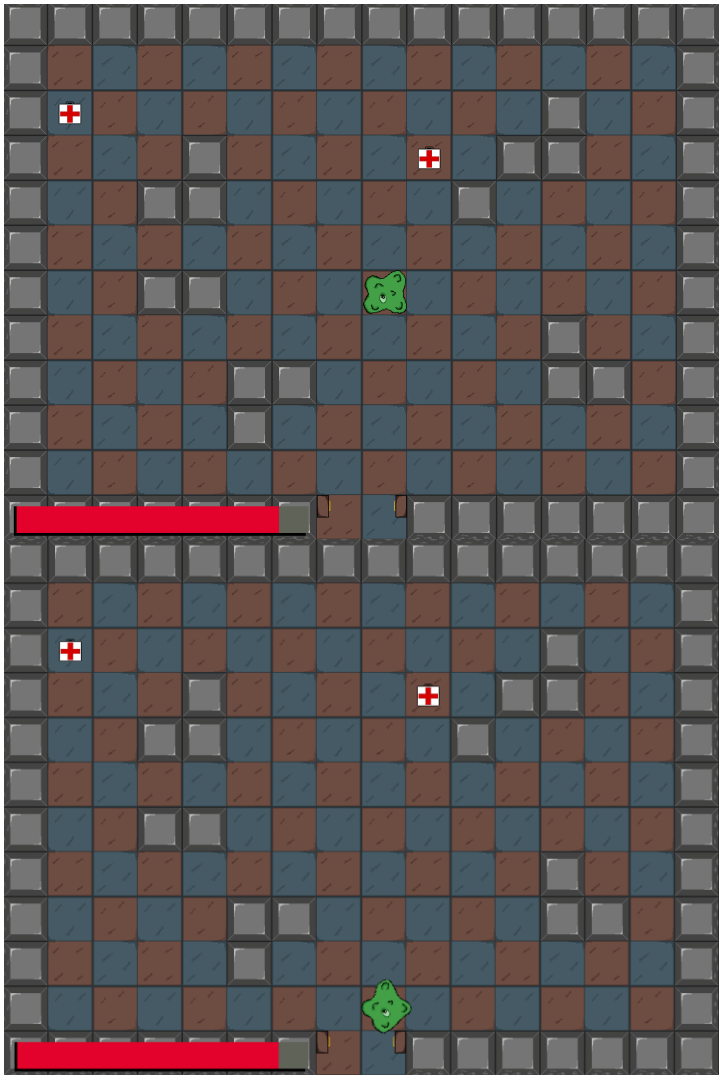


Fig 2.1.3
The player sprite moved left

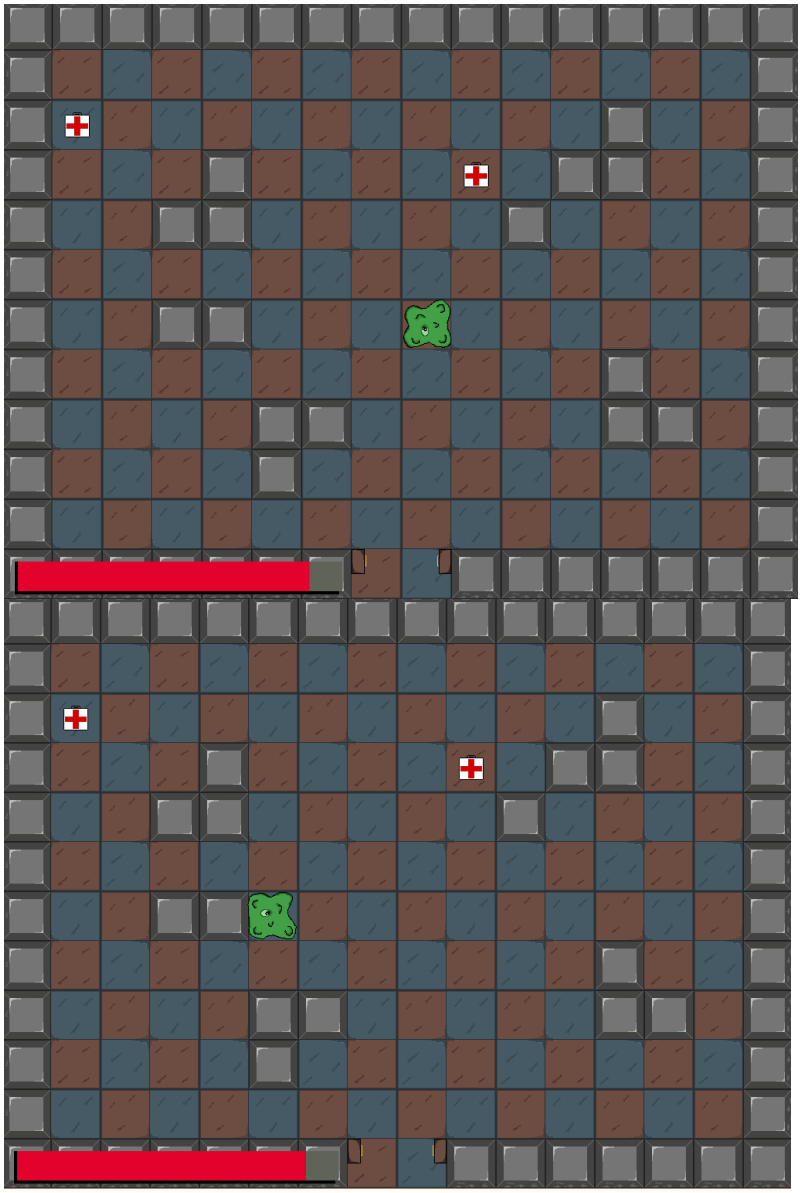


Fig 2.1.4
The player sprite moved right

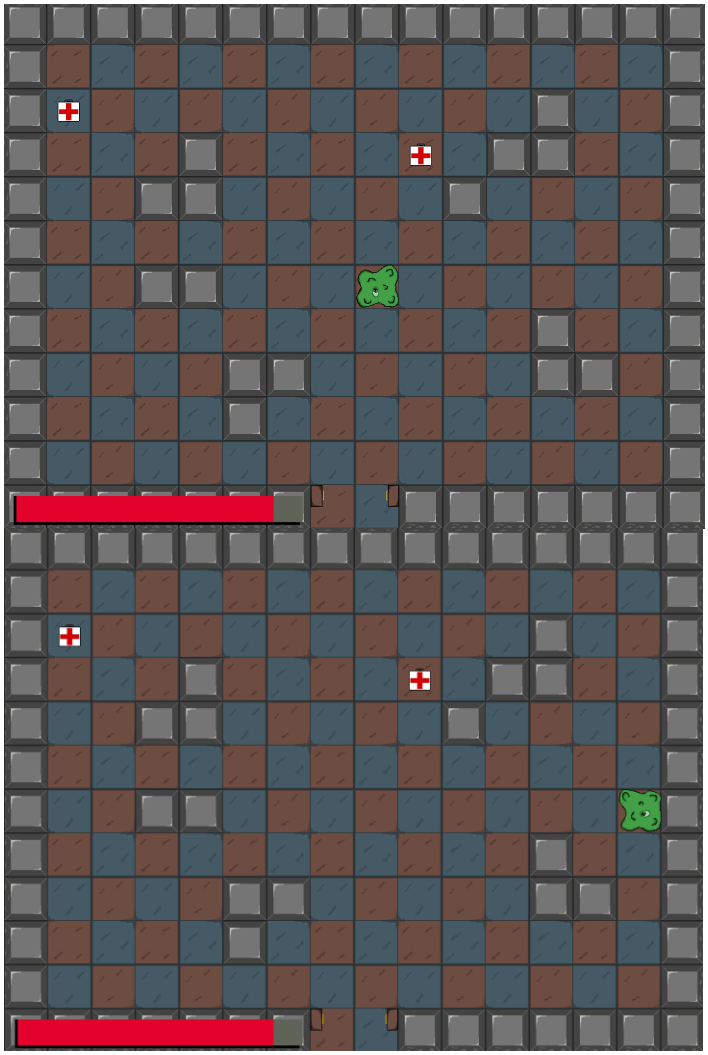


Fig 2.2.1
The player sprite moved diagonally to the top left

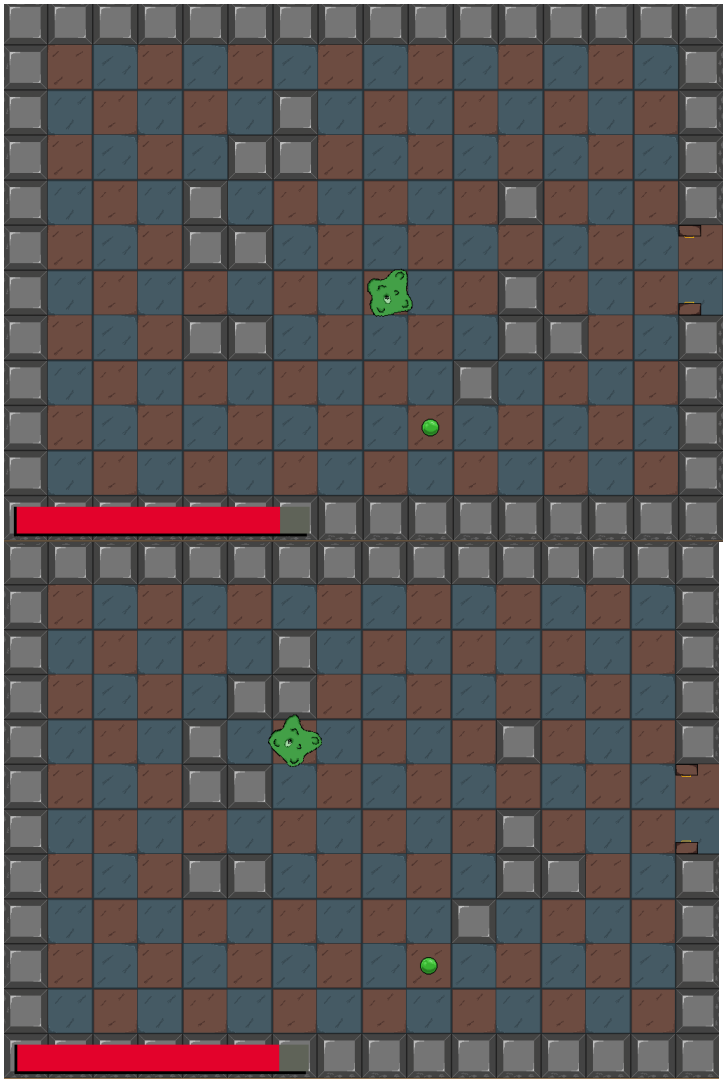


Fig 2.2.2
The player sprite moved diagonally to the top right

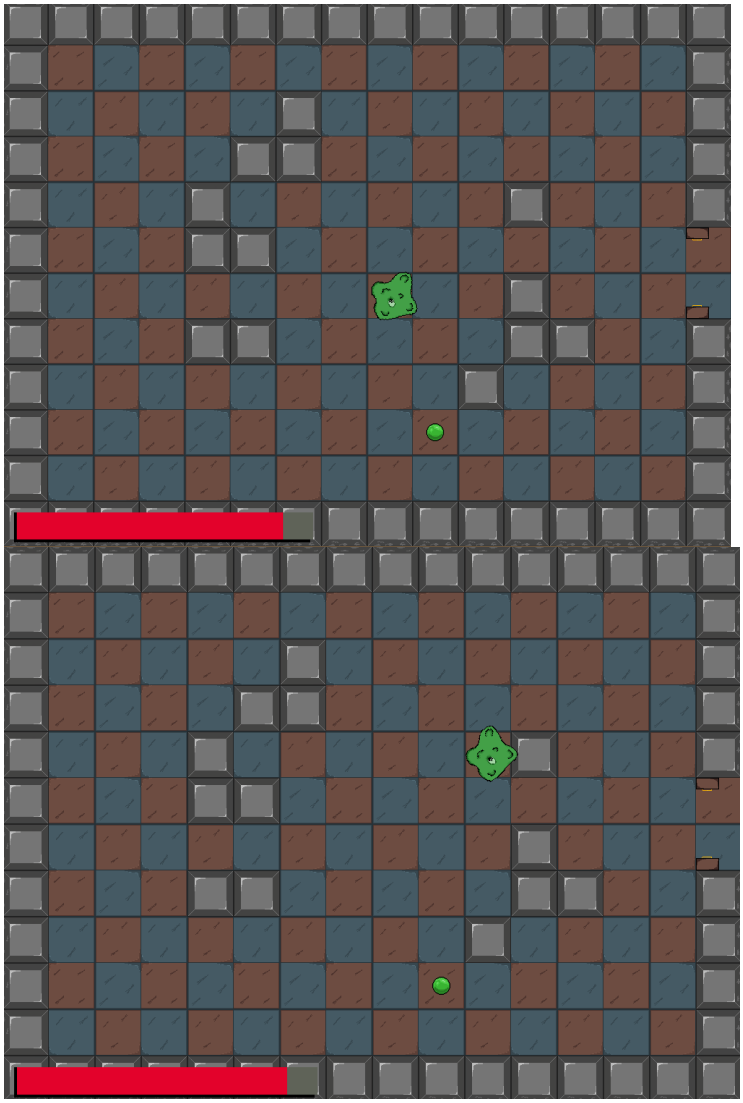


Fig 2.2.3
The player sprite moved diagonally to the bottom left

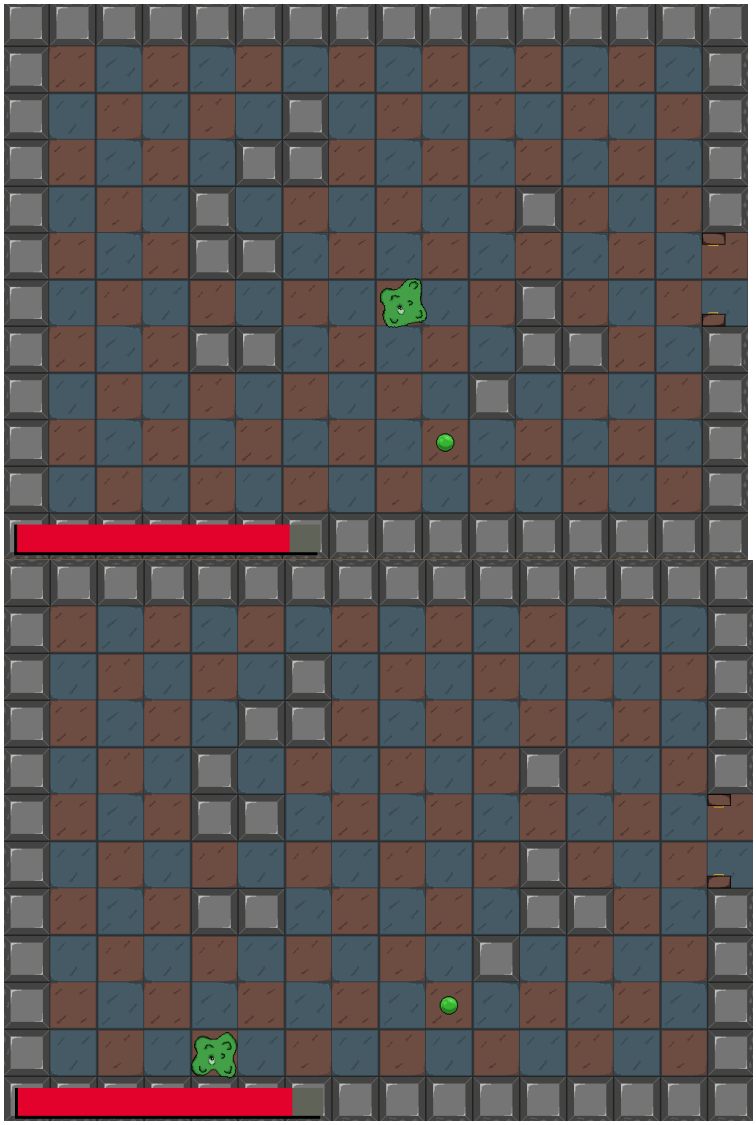


Fig 2.2.4
The player sprite moved diagonally to the bottom right

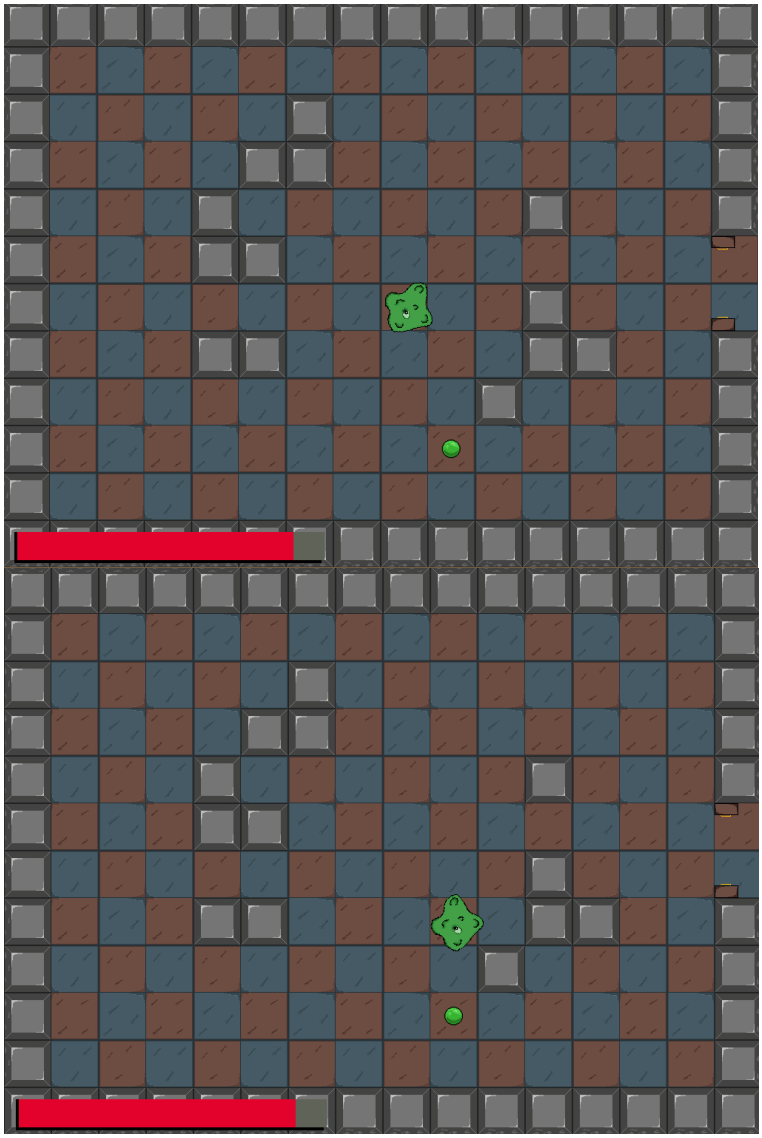


Fig 2.3

The player sprite looks right at 90 degrees towards the cursor (values printed out is the angle)

The player sprite looks at the bottom left 45 degrees towards the cursor (values printed out is the angle)

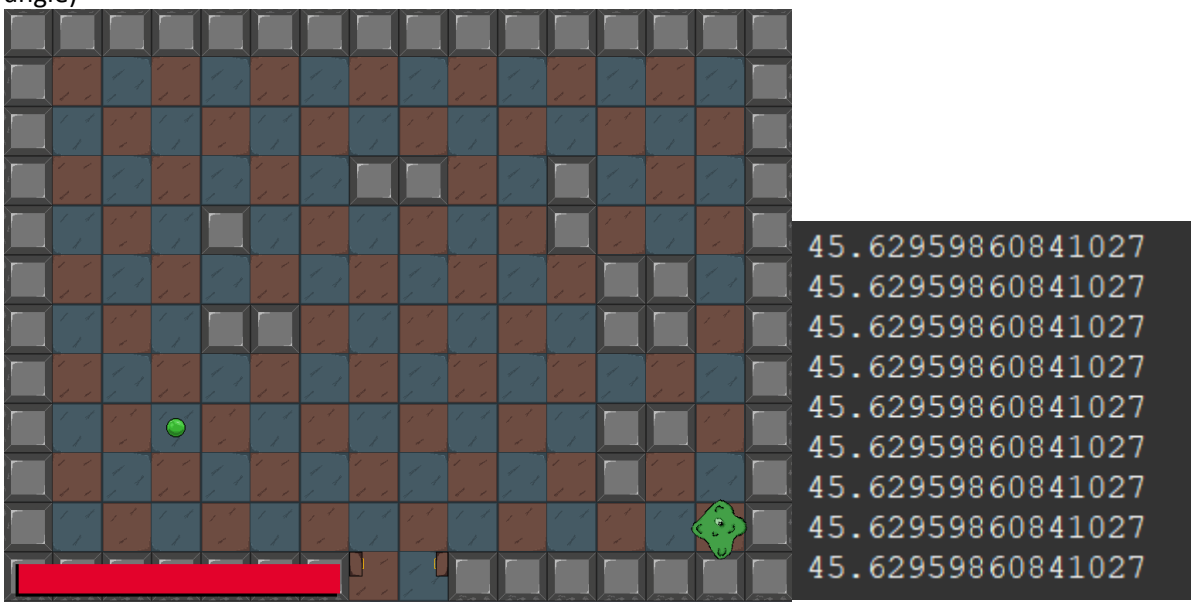


Fig 2.4
The red circle is where I clicked.

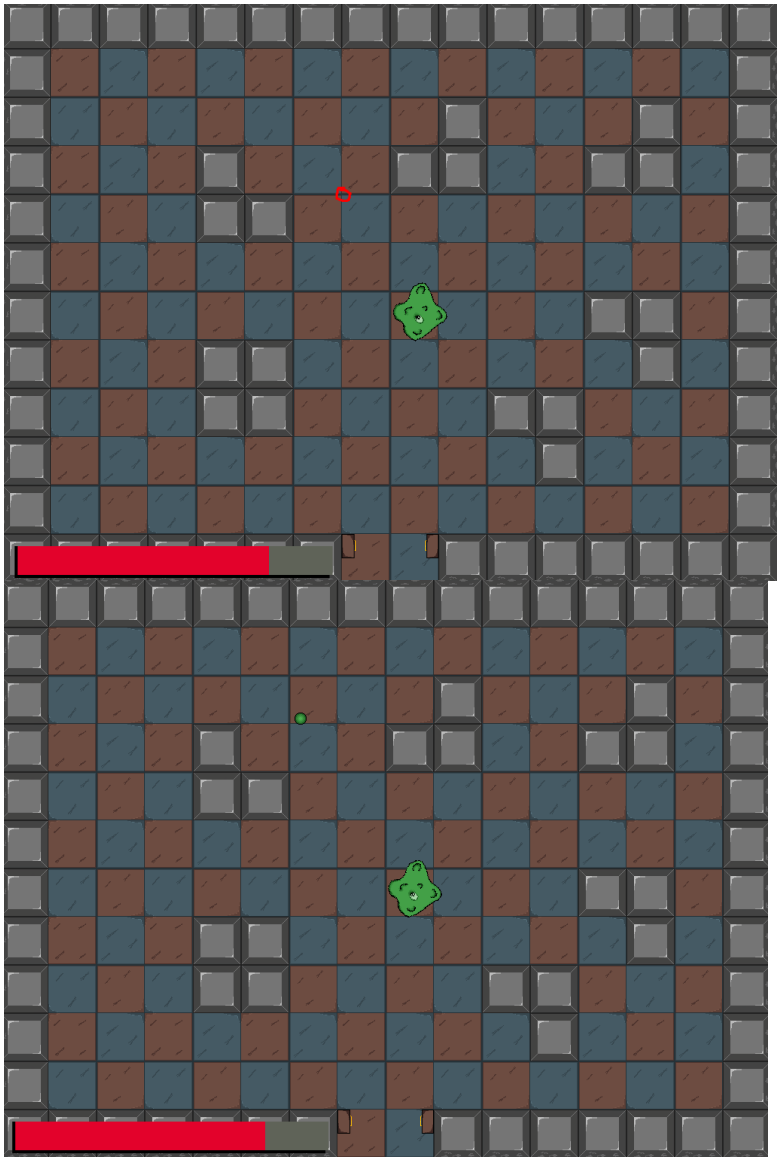


Fig 2.5
When the enemy is killed, the door opens.

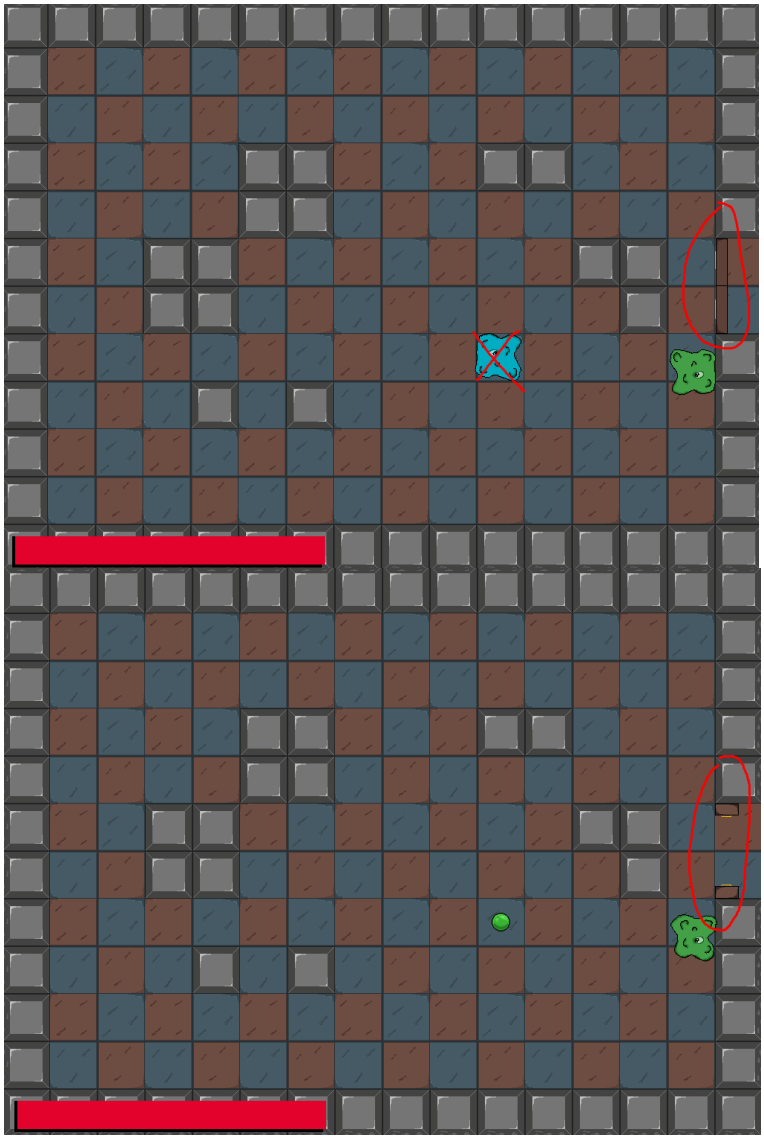
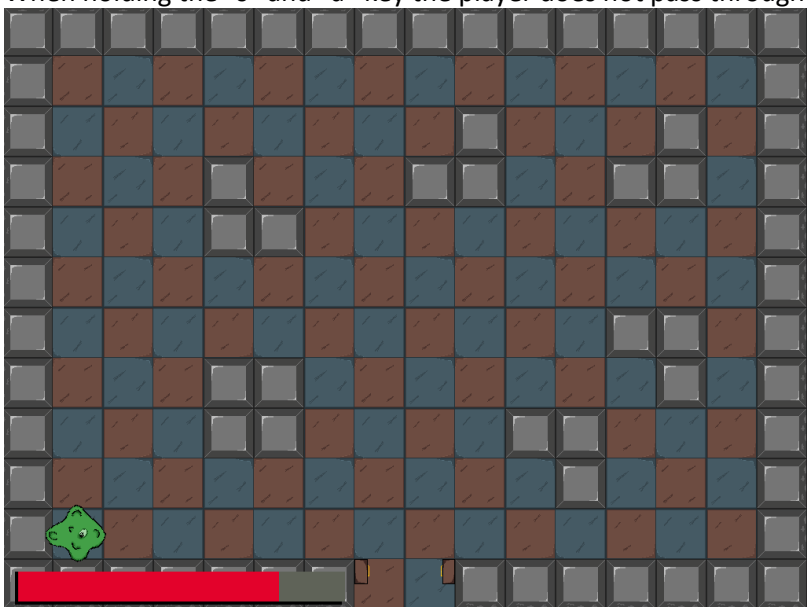


Fig 3.1

When holding the “s” and “a” key the player does not pass through the walls.



When holding the “w” and “d” key the player does not pass through the walls.

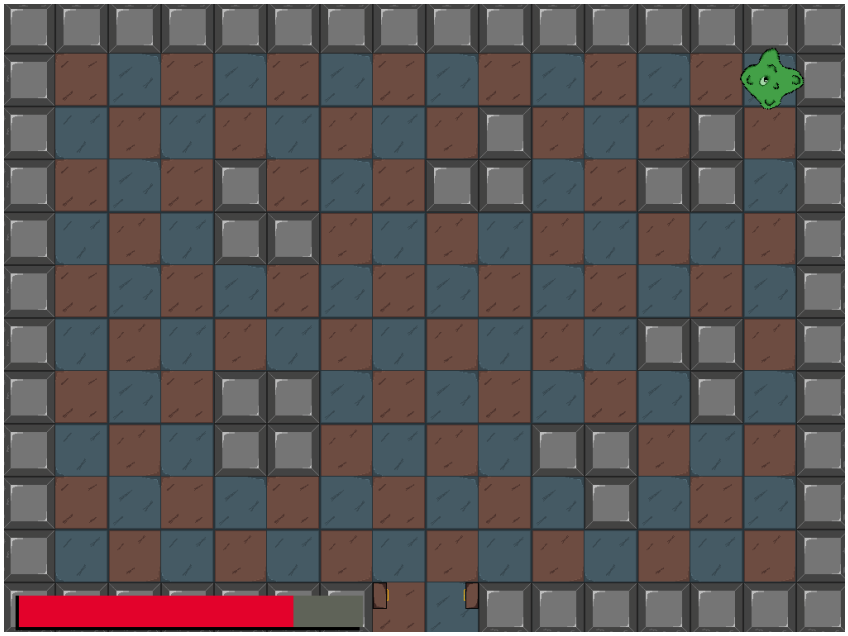
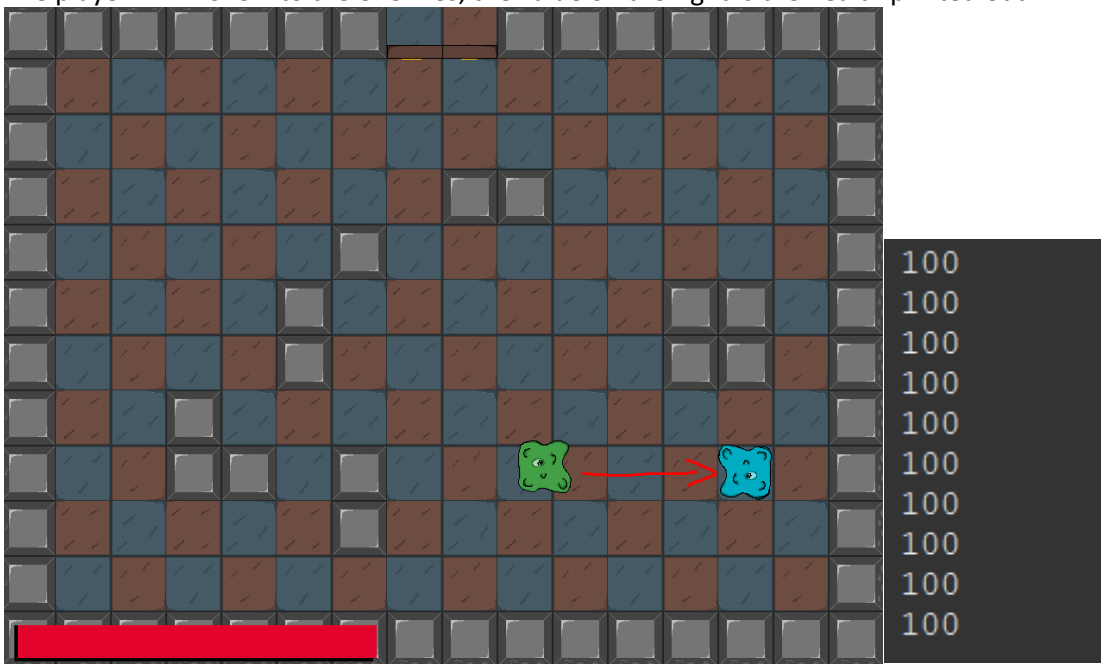


Fig 3.2

The player will move into the enemies, the value on the right is the health printed out.



When hit 10 health is taken off.

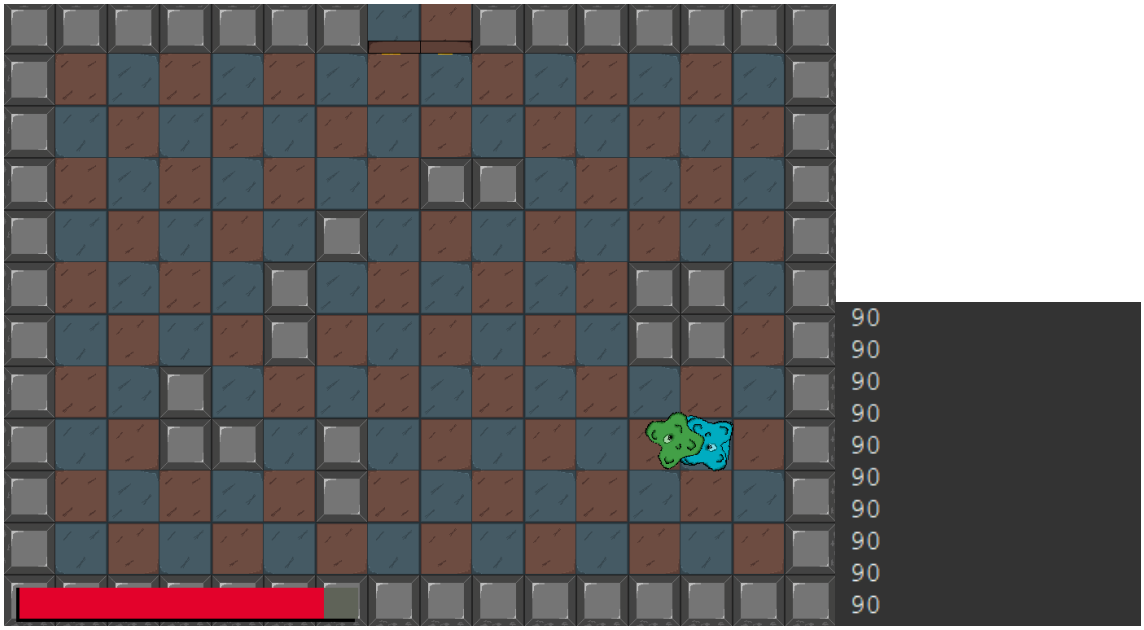


Fig 3.3
The player does transition to the next room.

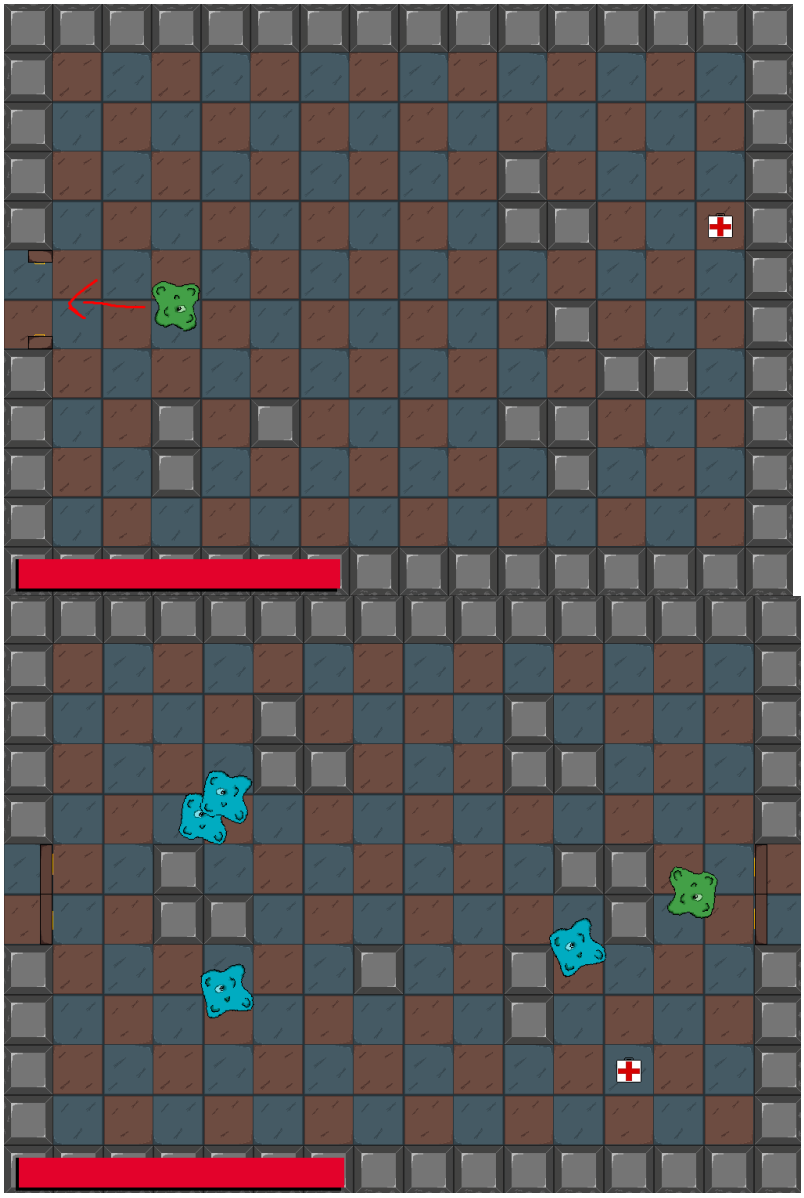
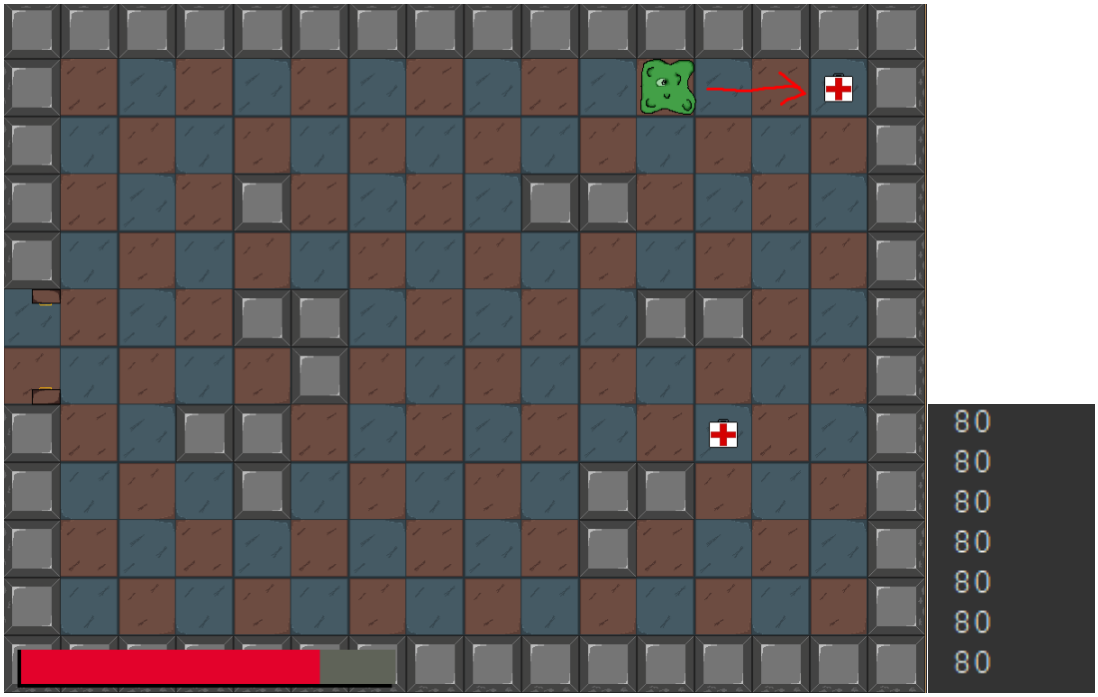


Fig 3.4
The player sprite will walk into the health pack.



The pack disappears and the health value is increased by 10.

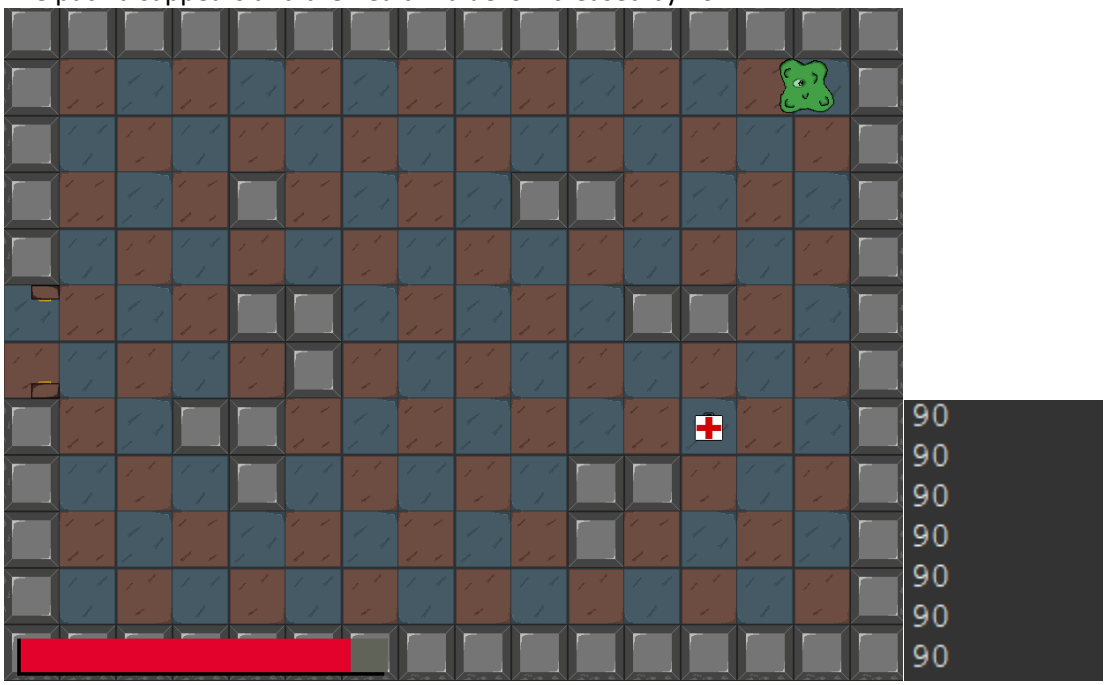


Fig 3.5
The red circle is where the player will shoot.

When the play shoots the enemies health decreases by 10 (value on the right)
Fig 3.6
The projectile hits the wall and the projectile disappears.

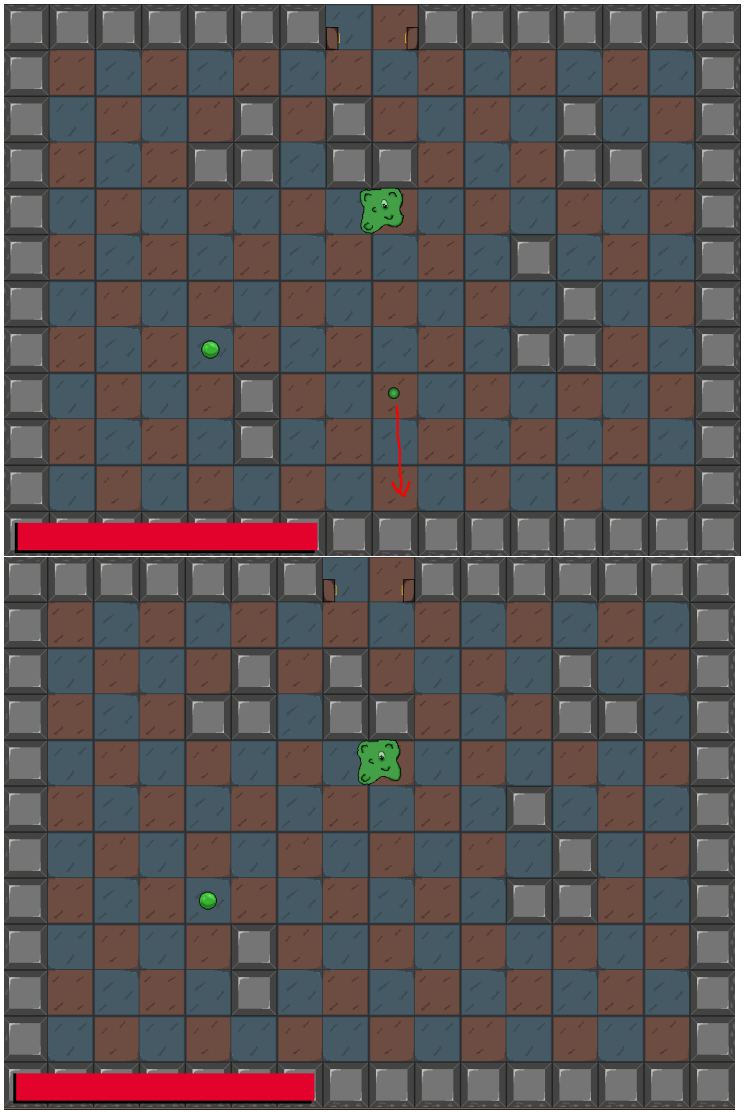
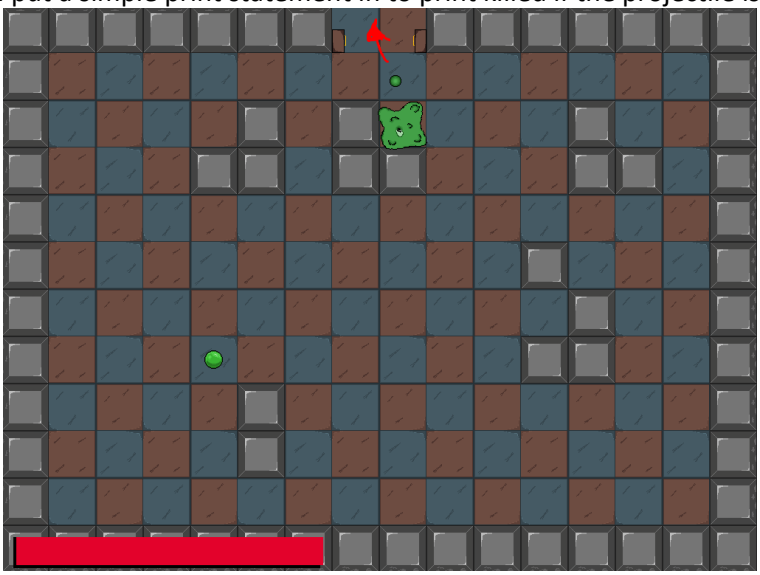


Fig 3.7

I put a simple print statement in to print killed if the projectile is destroyed.



```
if self.OffScreencheck() == True:
    self.kill()
    print("Killed")
```

```
Killed
```

Fig 3.8

The boss will bounce off the wall and collide with the player

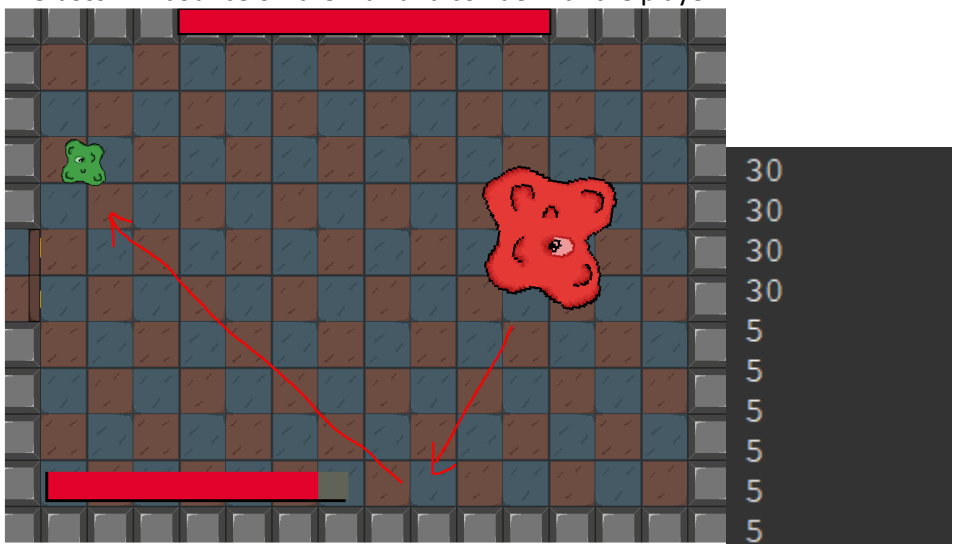


Fig 4.1

The red dot is the beacon and the enemies are path finding towards it. The text on the right is the print test to see whether the search is breadth first or A star search.

Fig 4.2

The text on the right is the print test to see whether the search is breadth first or A star search.

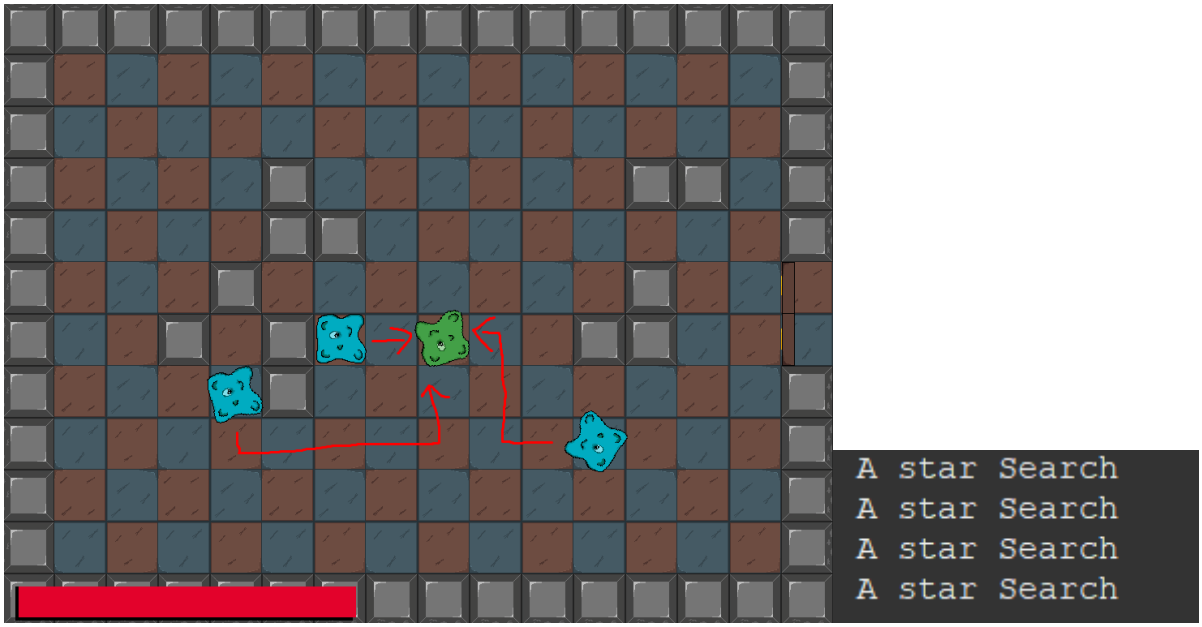
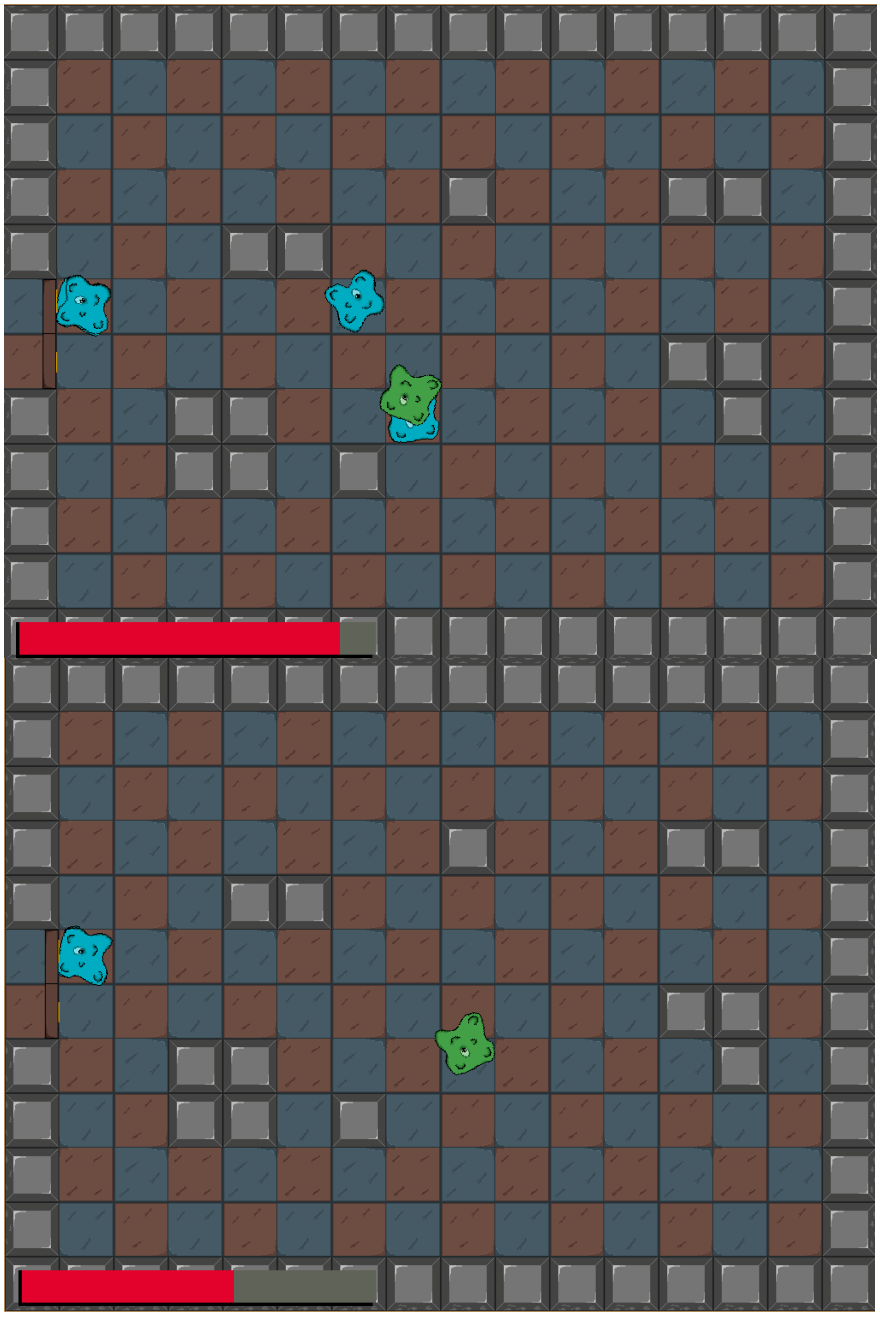


Fig 4.3
All the enemies stay in the goal node, unless it changes.

Fig 4.4
The enemies stop on the player and deal damage.



Erroneous Testing Screenshots

Fig E1.1

The division error pops up because the program is trying to divide by zero and that would be infinite.

```
    self.vx = (self.difx / self.dist) * self.vel
ZeroDivisionError: float division by zero

>>>
```

Fig E1.1

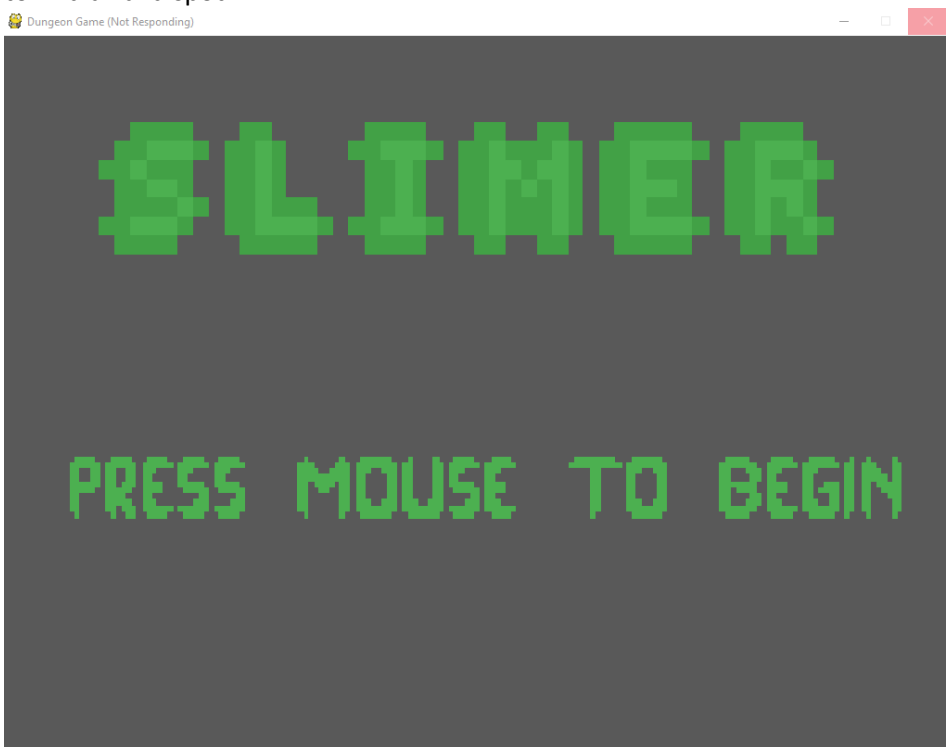
```
try:
    self.vx = (self.difx / self.dist) * self.vel
    self.vy = (self.dify / self.dist) * self.vel
    self.x -= self.vx
    self.y -= self.vy
    self.rect.center = (self.x, self.y)
except ZeroDivisionError:
    print("Destroyed")
    # // Except statment to catch a zero error in the case
    # self.distance is equal to zero.
    self.kill()
```

When clicked in the middle of the player sprite the test print statement prints out and the projectile is destroyed

```
Destroyed
```

Fig E2.1

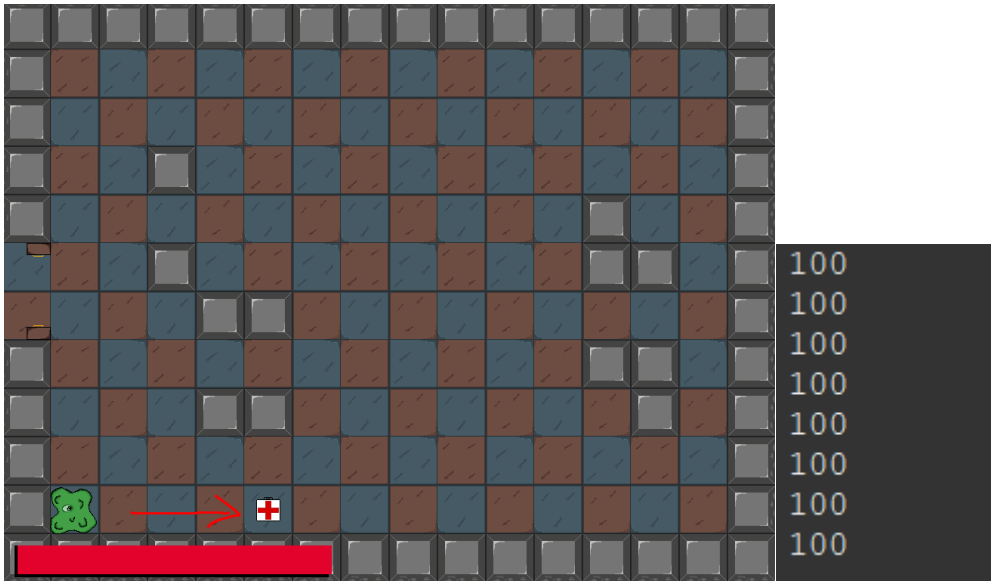
The program is stuck in a loop looking for a valid spot to place a wall, in which it is impossible to find a valid spot.



Boundary Testing Screenshots

Fig B1.1

The player sprite will collide with the health pack, the values are the health printed



The player's health does not increase past the max and the value does not increase.

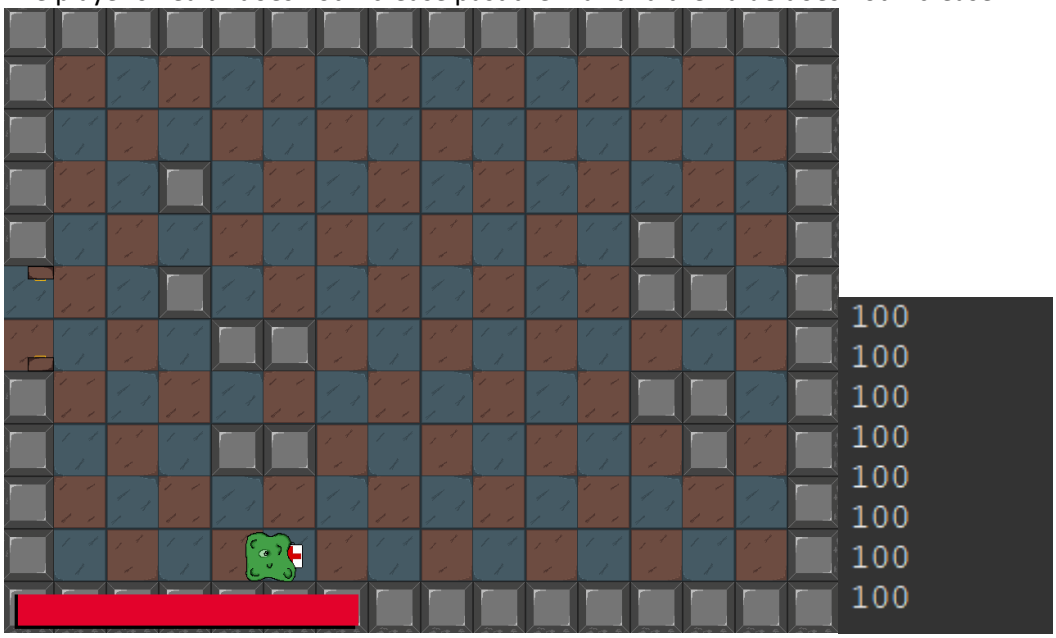
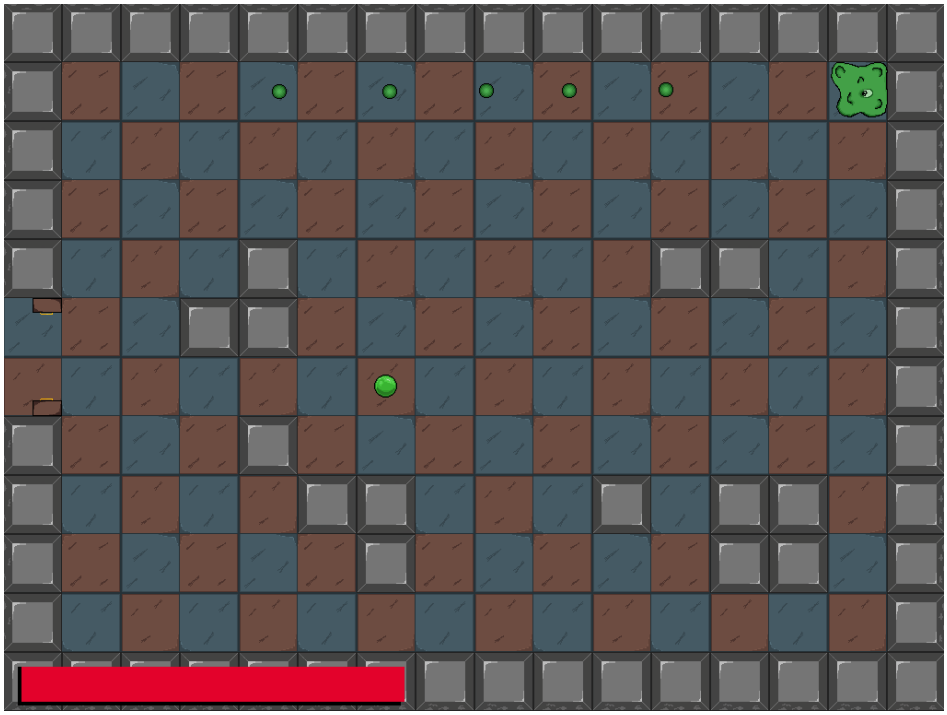


Fig B2.1



Evaluation

Original Objective Evaluation with client feedback

Aims and Objectives:

1. *Constructing the map so that it's different each time.*
    a. *Layout a map*
    b. *Add a wall entity which is placed all around the edge of the map, to act as a barrier for the room.*
    c. *Having the map be randomly generated, with wall entities in the middle of the room to give the player some cover and diversity to the layout, to provide a different experience for the user each time.*
    d. *Add a texture to the wall entity*

Objective 1

In my opinion, the objectives for this section have been for the most part, added to the game and function with how they were meant to. Furthermore, I could have made the map larger with a larger tile size to enable me to add more block types into the game to make the game feel completely different each time. This could also enable more complex algorithms to procedurally generate the rooms. Despite this, I feel that objective 1.d was done well and ensures that the user gets a different experience each time. Furthermore, my client agreed that all the objectives in this branch were met. My client liked that the map was generated differently each time and thought it gives the user a different experience each time. Although, he also said that it would've been better if the map had more places to explore with it being larger. As me and the client agreed in this point, the changes I would make in the future is to make the map larger so the player can move around more often.

2. *Add the main character sprite.*

a. *Make sure the movement of the character can be moved with the buttons WASD, also make sure that there's smooth movement when walking at the base speed in all directional axis.*
b. *Make the character sprite look in the direction which the mouse cursor is facing, allowing the player to move around in 360 degrees.*
c. *Make the object for the character have base Health, Attack damage and speed statistics*
d. *Add Main Sprite Texture.*
e. *Animate the sprite with several images.*

Objective 2

In my opinion, all objectives in this section were met and work how they were intended to. The character meets the objective 2.a well with the player being able to move in all eight directions and slide up against the wall if two buttons are pressed at the same time. The player sprite did not pose any major errors and moved though the map levels smoothly. My client thought that the objectives were fully met, and he said that the character movement felt smooth and the controls were easy to use. My client thought that the player didn't need any improvement and liked all the movement when the player interacts with walls within the map.

3. *Add a normal attack for the sprite.*
    a. *When the MOUSE1 button is clicked make sure that the character attacks and sends out a projectile.*
    b. *Make sure the projectile is shot out the way that the player is facing.*
    c. *Projectile must have a texture.*

Objective 3

In my opinion, all the objectives in this section were met and successfully work how they're supposed to. When MOUSE1 is clicked the character shoots a projectile ball in the direction its facing. Despite this, I could have made it, so the sprite also has a secondary attack that takes a longer time to charge up, but in the long run does more damage. My client thought that the projectiles were good, and that the mechanics were just what he wanted to see. My client also agrees that all the objectives were met.

4. *Make sure Collisions between the characters and map work.*
    a. *When a projectile is shot, make sure it is stopped by a wall and doesn't go through it.*
    b. *Make sure that the character and other entities cannot no-clip through the wall and or get out of the map in any way.*
    c. *Detection when a projectile hit an enemy, make sure the correct stats are subtracted.*
    d. *When an item is walked over, a collision should be detected and then do whatever that item is intended for.*

Objective 4

In my opinion, all the objectives in this section were met and all the collisions in the game worked well. Furthermore, I fell that trying to add improvement to already working collisions is a hard task, unless you are applying some sort of algorithm to it that benefits the game more. My client said that the collisions worked well on all fronts, due to them being an integral part of every game to reduce bugs. My client thought that all the objectives were met. My client also said he liked the addition of collisions between the open doors and the player sprite and said that there were smooth room-to-room transitions.

5. *Procedurally generate items within the maps.*
    a. *Make sure to design fair and balanced items.*
    b. *Make sure to randomly add items into the map*
    c. *When an item is dropped, make sure to correctly buff the player's stats.*
    d. *Add textures for different items.*

Objective 5

In my opinion, all the objectives for this section were met. My client also agrees with me, however, he said that he would like to see more items added into the game. For example, a larger projectile item, increased damage item and an item that can give you a shield before you die. I feel that these are good item ideas and if I was to make the game again, I would add these in to make the game more enjoyable. Despite this, the client said he thought I did objective 5.a extremely well with the health packs putting the player's health at chance.

6. *Addition of enemies with pathfinding capabilities.*
    a. *Have enemy objects suited with base stats like Health, Attack damage and Speed.*
    b. *Have an algorithm(s) in place to allow the enemies to path find towards the player to attack it when the collision happens.*

Objective 6

In my opinion, I think all the objectives are met, but also, I think that I have surpassed the objective 6.b by adding two types of path finding instead of one, to allow the game to have more diversity. My client agrees that all the objectives have been met, he also agrees that the addition of another path finding algorithm, for the item, is a great addition. Furthermore, my client feels that the enemy's health and stats are at the right difficulty to be able to have fun with the game.

7. *Addition of End Boss Level.*
    a. *Implementation of boss.*
    b. *Make sure the boss is suited with its own base stats and special attacks.*
    c. *Make sure that each boss has a texture,*
    d. *Once the boss has been defeated, display game over.*

Objective 7

In my opinion, all the objects for this boss section have been met. My client also agrees with this statement. He also said he likes that the boss has a harder level of difficulty than the other rooms, which adds more hecticness. Despite this, my client also said that he would like to see the addition of more bosses within the game with different attack stages. If I was going to make the game again, I would use more of the boss ideas from the design section and implement them into the game.

Future Development

Current problems

The current problems are that the program can drop some frames when pathfinding a complex path, despite this, the performance isn't hindered too much, so the problem isn't massive, but could use some attentions to make the path finding algorithm even more efficient.

Improvements

For future improvements, I would take my clients advice and add more types of enemies/bosses into the game. I feel this would give the game more excitement when playing due to the combinations of enemies. Despite that, I would also like to make the map larger, so there's more areas to explore. Furthermore, If I had more time for the project, I would also like to work on the network side of things and try to make the game a multiplayer experience, so you are able to play it with your friends.

On the other hand, If I had more time, I would also add a "create your own map" section where the user can create their own map and send it to other people to see if others can try to beat it.

Finally, it will be very useful to see what health the non-boss enemies are on so the users can strategies and play the game without trying to guess what the enemy's health is on. I believe that this will be a good implementation for the beginner players who have no idea what the game it about.

Conclusion

Overall, I am pleased with the approach of making my application. I feel that, overall, I have surpassed the objectives I set myself before programming my project. I am confident that my game works well with little to none bugs in. Despite this, my client is happy with the final outcome and enjoys the game. On the other hand, if I had more time, I would try to implement all the improvements above as I think it would make the game a lot more enjoyable. Finally, I am pleased with all the program and hope to eventually carry the game past A-level and work on it some more.

Appendix – Client Confirmation