

Manhunt

Name: Muqtasid Zayyan Dar

Candidate Number: 2905

Centre Number: 33413

Analysis	5
Problem Identification:	5
What is Manhunt?	5
Why have I decided to create this?.....	5
Where computational techniques can be used to solve the problem	6
Stakeholders	8
Research.....	10
Escape the Ayowoki	10
Mark of the Ninja	12
Metal Gear Solid 1.....	14
Proposed Solution.....	16
Essential Features	16
Limitations:	17
Project Requirements:	18
Success Criteria	19
Input:.....	19
Process:.....	19
Output:.....	23
Aesthetic:	24
Design.....	26
Decomposing the problem	26
4 Screens:.....	27
Hunter:.....	29
Player:	39
Map:	46
Physics:.....	49
Round Ends:	54
Describe the solution:.....	56
Extra variables:.....	56
Classes.....	56
Pseudocode:.....	80
Usability:	127
Approach to testing:	139
In Development Testing:.....	139

4 Screens:.....	139
Hunter:.....	139
Player:	140
Map:.....	141
Physics:.....	141
Round Ends:	142
Post Development Testing:.....	142
Development.....	147
Version 1:.....	147
Version 1.1:.....	147
Version 1.2:	152
Version 1.3:	155
Version 1.4:	165
Version 2:.....	169
Version 2.1:	169
Version 2.2:	170
Version 3:.....	179
Version 3.1:	179
Version 3.2:	182
Version 3.3:	186
Version 3.4:	188
Version 4:.....	194
Version 4.1:	194
Version 4.2:	198
Version 4.3:	200
Version 4.4:	202
Version 5:.....	205
Version 5.1:	205
Version 5.2:	208
Version 5.3:	217
Version 5.4:	223
Version 5.5:	232
Version 5.6:	237
Version 5.7:	246
Version 6:.....	251
Version 6.1:	251

Version 6.2:	264
Version 6.3:	269
Version 6.4:	277
Version 6.5:	283
Evaluative Testing	290
Black Box testing table:	290
Stakeholder Testing:	323
Evaluation:	330
Success Criteria Evaluation:	330
Usability Features:	339
Maintenance	340
Further Developments	343
Limitations	345
Development Style:	346
Code Listing	347
File: main	347
File: colours	352
File: hunter	352
File: mapObjects	367
File: maps	370
File: objects	376
File: physics	382
File: player	390
File: screens	400

Analysis

Problem Identification:

What is Manhunt?

For my NEA computing project, I am going to create a top-down stealth game called Manhunt. A stealth game is a game in which the player must avoid or run from some sort of hunter. In my game the hunter will be an AI/bot which will be searching for and chasing the player through the map. The player needs to either complete objectives to unlock the exit/reach an endpoint in the level. However, the hunter can 'hear' and 'see' the player if the hunter is close enough. If the hunter sees the player then the hunter will chase the player until the hunter has either caught the player or the player has escaped the hunter's vision for more than a few seconds. When the player makes too much noise and the hunter is close enough to hear it then the hunter can decide to go in the direction of the noise to search for the player. The player can also use this to their advantage by picking up objects around the map and then 'throwing' the object in a certain place to make a noise there so that the hunter gets distracted. The player can also 'hide' in certain places to avoid being caught by the hunter – however, if the hunter sees the player 'hide' then the hunter can still catch the player. The hunter will look completely different to the player so that it is easy to identify them as the hunter. The hunter will be randomly traveling around the map while it has not seen or heard the player

Why have I decided to create this?

I have decided to create the stealth game Manhunt as I believe that recently, there are less fun stealth games being created and the stealth genre is also dying out with not as many big games being primarily focused on stealth. Furthermore, many people are also limited by the minimum specifications that the game needs and cannot play it as their system may not be able to run it as smoothly as a higher end system which they cannot afford. Therefore, I have decided to create this game as I want to create a fun stealth game that anyone and everyone can play whenever and wherever, without having to think twice about whether they have enough storage space for their game or whether their graphics processing is good enough to run the game. Most stealth games nowadays are also aimed at an older audience which means that most kids can not play the game either (although many do not check the age rating and play the game anyway). So I have decided to create this game so that more children have access to the stealth genre as well. The only genre that is similar to the stealth genre that has been gaining popularity recently is the survival horror genre which – although contains elements of stealth – is still quite different form the usual stealth genre.

Where computational techniques can be used to solve the problem

Computational Techniques	Explanation	Justification
Path Finding	The hunter in the game must search for the player throughout the level – either through random routes if the hunter doesn't know the possible location of the player or a direct route if the hunter knows where the player is. The player can also cause a distraction by using certain objects found throughout the level.	This will make the game more interesting and tense as the player will not be able to predict the hunter's path through the level which will also make it harder escape the level. When the hunter 'hears' the player (this is a situation where the hunter knows where the player is) the computer needs to calculate the shortest route to the origin of the sound that the hunter has 'heard'. When the hunter doesn't know where the player is then the computer needs to randomly generate paths for the hunter to follow so that the player cannot predict where the hunter is going to go.
Sound distribution (Physics)	As the hunter has the ability to hear the player – it needs to be somewhat realistic so that the hunter cannot hear the player if the hunter is too far away. Furthermore, if the hunter is further away the location of the player should be more vague for the hunter.	Some stealth games mainly revolve around the seeing aspect of the hunter and, therefore, by making hearing also a big part of my stealth game, it makes it more interesting. A computer will have to check the location of the hunter in relation to the player and calculate if the hunter would be able to hear the player – this would include things such as number of walls between the player and the hunter and the distance away if there is empty space between them. If the hunter is far away but is still able to hear the player then the computer needs to calculate a vague location for the hunter to go to.
Field of view calculation (Physics)	As the hunter also has a field of view, the hunter should not be able to see the player through walls but should be able to see the player from the opposite side of a long corridor, if no objects are in the way and as long as the player is not hiding.	If the hunter can see the player no matter where they are on the map the game will not be as intense as the player will know that the hunter is always behind them but by having the hunter not knowing where the player is it makes it more ambiguous for the player. A computer will have draw lines from the hunter which would represent the hunter's field of view and then use that so that if a line is colliding with an object then the hunter can only see until that object and not through/ past it.
Collision (Physics)	The game needs to be somewhat realistic in the sense that the player and hunter can not just magically run through the walls or other objects	The computer will have to

AI	Hunter is an NPC – Non Player Character and needs to decide when they have heard the player, when they have seen the player, what paths to use around the level etc.	This will add an element of realism to the game, although not incredibly human-like, allowing the computer to decide in certain situations makes it even more unpredictable and increases the tension within the game. The computer will constantly check and update information on its surroundings using iteration and then adapt to the changes to search for the player in the level.
Classes/ Objects	Within my game things such as the walls, floors and other objects that will be placed throughout the level will be used multiple times	A class will be more efficient because whenever an object needs to be created then the class and the method can be called upon rather than rewriting the code each time you want to create an object. Furthermore, this will be useful if the end product generates a random level each time
Simple and smooth controls	In order for the player to move the character around, through the level on their screen they need to be able to use a button or something similar which will move the character in that direction.	It will be easier for the player to learn how to play the game if there are simple controls for things such as moving throughout the level, and these controls need to operate smoothly so that the game feels refined for the player

Stakeholders

Persona:

My game will mainly be targeted at kids and teens, so around the ages of 8 – 17, as anyone younger than 8 may find the game harder to play since it may be too complex for them and for people over the age of 17, they may find the game too simplistic and boring. The target audience would also have some previous gaming experience as this may help them to learn how to play the game quicker. Furthermore, my persona should have almost no experience in stealth games as my game is designed to be more of an introductory game to the stealth genre rather than a leading competitor. As my persona will be people who are relatively new to the stealth genre, my game will have a more simplistic design so that the aims of the game and how to play it will be easy to understand even for non-experienced gamers. Moreover, as my game is targeted at somewhat non-experienced gamers, the specifications of a computer needed to run my game will be low so that anyone can access it and play the game.

Stakeholder 1:

Name: Sameer

Age: 17

Suitability: He is an experienced gamer as he has played many games, however, he finds that modern games nowadays are quite monotone and is looking for a more unique game with interesting features, furthermore, he also thinks that modern games are too easy and wants something that is more challenging and can push him to his limits. In terms of ICT skill or experience, although he plays many games, he does not have software experience and doesn't have that much hardware experience. In my project he will be one of the testers of my game and will check and inform me of any bugs and will also advise me on what I could do to make the game more fun or interesting.

In order to facilitate his needs, I am going to make my game more interesting by adding features which are unique to the game and are not found in other games, an example of this is that it is a 2D top down game in which the hunter AI can hear which is not present in many games. In order to make the game more challenging, I am going to implement different modes for the hunter so that he can choose if he wants a harder mode. Furthermore, I have made the game more unique by merging parts of the horror genre into my game as well – this will make the game more interesting for him as it will be a combination of different genres of gaming in one game.

Stakeholder 2:

Name: Shahir

Age: 12

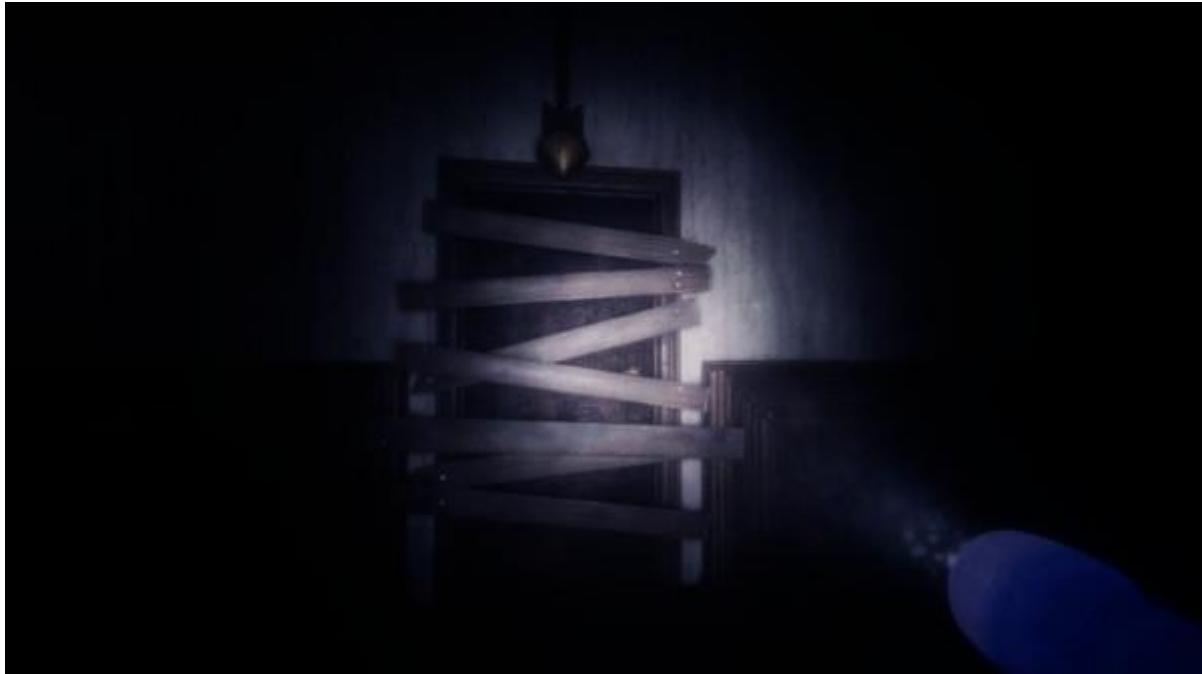
He fits into my target persona as he is within the age range and is a frequent gamer, he hasn't played a game in the stealth genre before and is looking for a game that can introduce him to this genre which is unique and fun to play. As he is a frequent gamer, he has relative experience in playing games, however, he does not have any software or hardware skills and may not be able to assist in these areas. Therefore, he is mainly here to assist in the content of my game and will be able to advise me on different aspects of the game that he enjoys and others aspects that may not be needed or need to be changed.

In order to facilitate his needs, I am adding different modes into the game so that if a mode is too hard for him then he can easily choose a different mode which is easier. Furthermore, I have incorporated multiple elements of the stealth genre into this game such as the limited field of view for the player or the hunter's ability to chase the player when seen. Also, in order for him to be able to quickly get familiar with the controls, I will add a screen which will display the controls so that he can quickly learn the controls for the player and be able to play the game.

Research

As my game is a relatively new concept in the 2D top-down dimension, there are not many other games that exist which are remarkably similar so the games I have found each have some aspect of my game that I want to include. As there are very few 2D games in this genre, most of the games that I will be researching will be 3D games although this will not be the case in my game.

Escape the Ayuwoki



In this game the character you play is someone who has been captured by the 'ayuwoki' and is trapped in his mansion. The objective of the player is to escape the mansion without being caught by the ayuwoki and the ayuwoki can hear the player's microphone and movements if it is close enough. In order to escape the player must find the key to unlock the main door to the mansion in order to escape. The ayuwoki is an AI that can hear the player but cannot see the player so in order to make sure that the ayuwoki does not catch the player, the player needs to remain quiet – both in real life and in the game. If the ayuwoki is close enough to hear the player and the player makes a noise, then the ayuwoki will move towards the players location; when the ayuwoki is nearby then the player must remain quiet by not speaking into the mic and remaining still in the game so that the player is not caught by the ayuwoki. If the player speaks into the mic loud enough that the ayuwoki can hear it or moves then the ayuwoki will capture the player and the player will lose.

Positives:

- Very good hearing mechanics – The ayuwoki can hear the player throughout the game if they are moving and the faster they are moving the more sound it makes, furthermore, even the players own voice can be heard by the ayuwoki (A1)
- Ayuwoki has a good AI – It is able to figure out where the sound of the player came from and goes towards that sound and checks it and if the player makes another sound it is able to decide to capture the player if it is close enough (A2)

- Ayuwoki has good pathfinding – It goes throughout the whole mansion randomly so the player cannot predict where it will be which makes it quite tense and more interesting (A3)
- Can use the shift button on the keyboard to so that you character runs through the game quicker (A11)

Negatives:

- If the player is not a very loud person and they are able to keep their hands off the keyboard then it is quite easy to escape the ayuwoki each time you encounter since it cannot see, even with the unpredictability of its movements – This makes the game easy for some people (N1)

Conclusion:

Although the hearing mechanic is an important and unique part of my game, I will not make it as realistic and complex as it is in Escape the Ayuwoki as I think it will be too complex and unnecessary to monitor the players mic and make the AI hunter react to that. Furthermore, from researching this game, monitoring the mic so that the AI can react to that can be a disadvantage (N1) and it will take up unnecessary time to code. Therefore, this will not be included in my Manhunt game. However, the pathfinding algorithm (P1) within this game is quite good as it is unpredictable yet can quickly find paths to the player if they make a noise (P2), therefore, I will try and replicate this within my own game.

Mark of the Ninja



In this game the character you play as a nameless ninja and there is conflict between ancient ninja tradition and modern technology. It is a 2D side scrolling stealth/platforming game. As the ninja, the player must go through each level, without being noticed while killing any guards that stand in the way. While your going through the level you must be stealth and avoid guards where possible – this can be done by hiding behind a plant pot or in a doorway while the guards are walking past. Furthermore, if your sprinting then each step you take generates a loud footstep which is represented by a large circle of sound that originates from the player and can be heard by the guards if they are within that circle which will alert the guards to your location. The guards have a field of view (FOV) that if the player is in then they can see the player; if the player is caught by the guards then they lose and have to restart the level. It is easier for the guards to see the player if the player is in the light instead of the dark. The game has various weapons that the player can obtain.

throughout the levels which they can use to kill the guards. The game also has various platforming mechanics such as wall jumping and the player also gets a grappling hook in the first level which they can use.

Positives:

- Sound distribution – Although its probably not a perfect circle in real life, the distribution of sound originating from the player is quite realistic (A4)
- Sound mechanics – Most things can be heard by the guards when the player is not sneaking around such as the opening and closing of vents or the footsteps of the player (A5)
- Hiding spots – The placements of the hiding places are good since they are almost always in places where players need them (A6)
-

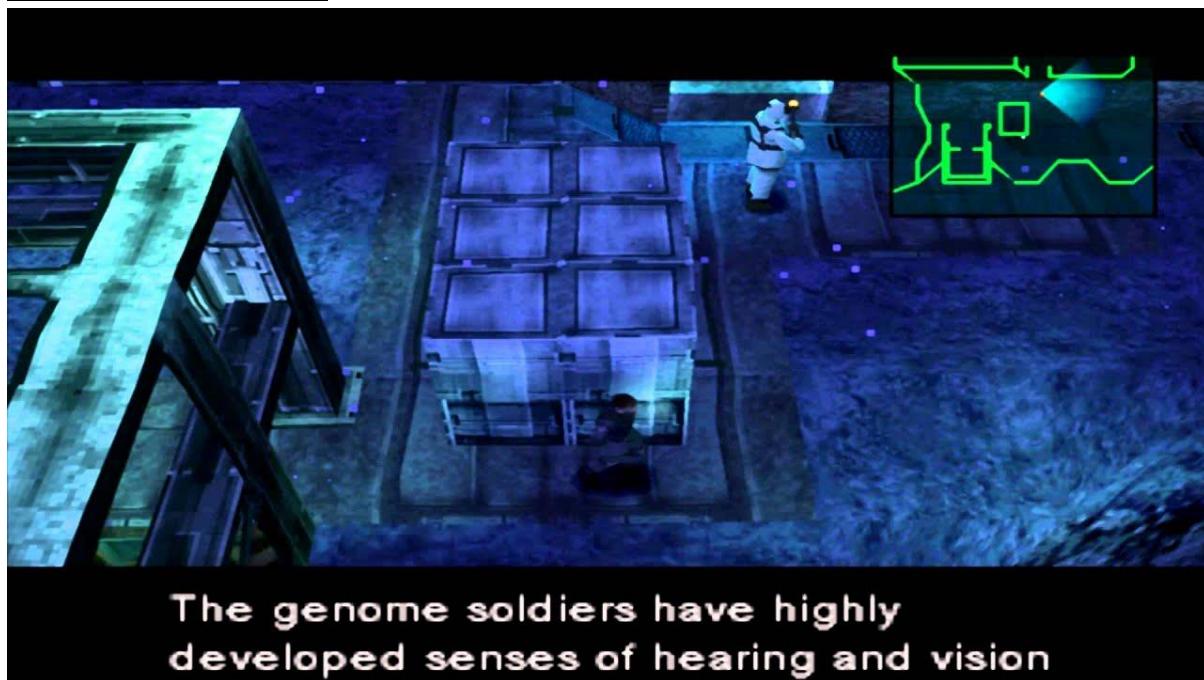
Negatives:

- The guard AI – Not very smart, it can make the game more boring as it is too easy to get past some of the main obstacles on the level. (N2)
- The guard FOV - too small – not accurate, the player can stand about half a metre away from the guard and they will still be out of the guards FOV(looks like they all need to go to Specsavers) (N3)
- 2D side scrolling – I believe that this adds to the predictability of the guards since the guards can only go 2 ways if they are not stationary (N4)

Conclusion:

Within this game the sound mechanics are very good compared to other games, when the player is sneaking around the game, most actions performed by the character that you would expect to make a noticeable noise do make a noise which can be heard by the AI guards (P4 & P5). Therefore, I would like to implement something like this within my own game as well. However, with regards to the AI, I believe that they are too simplistic and unrealistic as the guards are very predictable and have designated paths which they follow (unless they hear a sound originating from the player) (N2). Furthermore, the FOV of the guards is far too small and unrealistic as when the player seems like they are about a metre away from the guards, the guards still do not detect the player which is why I will be using a longer FOV as this will be more realistic. Within my own game I am planning to make it with top-down 2D graphics, and this game reinforces this as 2D side scrolling makes the game easier (N4). Furthermore, the 2D side scrolling means that it is easy for the player to escape the guards if they just move up a level since the guards do not have any platforming mechanics. Moreover, it makes the guards more predictable since they can only come from the right hand side, therefore, making the game less interesting and makes it somewhat repetitive.

Metal Gear Solid 1



In this game you play as the protagonist called solid snake and it is a 3D game. The player has to go through each 'level' and make sure they are not spotted by any of the guards, cameras, or lights. The player has a minimap in the top right corner of the screen which they can use to look for enemies and it displays the enemy's FOV so that the player knows whereabout the guard/enemies can see. The game is played in 3rd person but the player can also switch to first person in order to peek around corners to check for enemies. The enemies are played by a bot which takes set routes around the level. If the enemy sees the player then they will start chasing the player to where they last saw them and when they get close enough they will start shooting the player and the player can fight back and knock out the enemies to stop the enemies from chasing and shooting them. After an enemy has seen the player then there is an alert timer in the top right which tells the player how long they have to hide for until the enemies will stop being suspicious and chasing the player. The player has a health bar and can be shot a few times, however, once the player has run out of health then the player has lost and a game over screen is displayed.

Positives:

- Quite realistic field of view – It is a lot longer and slightly narrower than the FOV of the guards in mark of the ninja (A7)
- Has 2 different viewing modes for the player so that the player is able to see their surroundings clearly (A8)
- The player can crouch or go prone to hide under things.(A9)
- Player also has an FOV similar to the enemies (A10)

Negatives:

- There are no hearing mechanics so the player could just run around the level and as long as the enemies don't see the player, the player will not be caught (N5)
- The enemies are quite slow so the player can escape very easily even though the enemies have seen them (N6)

- The enemies' AI is not the best as they only follow a set route, so it is quite easy for the player to predict the movements and therefore, makes the game less interesting (N7)

Conclusion:

Although there are newer iterations of this game, they are more story like, have varying stealth mechanics throughout the different parts of the game and are far too complex to compare to my intended game. As this game is 3D they can implement 2 different viewing modes which immerse the player more (P8), although this is an advantage of the game, it would be too complex to code and would take too much time as I would have to consider translating a 2D landscape into a 3D landscape. The main aspect of the game I would like to try to implement within my game would be the sight mechanics in the game which is the FOV and how the AI reacts after they have seen the player (P7 & P10). This is because I believe that the FOV is the most realistic out of the 2D games and so I want to replicate this realism in my own game. One part of this game I am going to make sure that I do not include is the enemies AI or pathfinding as they always just follow the same route making them very predictable if the player looks for a pattern(N7). Also, after researching this game it has reinforced the idea of adding a hearing mechanics in the game as, without hearing, it is quite easy to avoid the enemies if you are not in their FOV (N5). Additionally, the enemies are a lot slower than the player in terms of moving speed which makes it easier for the player to escape from them and so within my game I will try and match the hunter and players sprint speed so that it makes the game more intense as the player will feel like the hunter is fast enough to catch them (N6).

Proposed Solution

These are the essential features that I am planning on implementing into my game along with a justification for each feature and a computational solution.

Essential Features

Feature	Justification	Computational Solution
Pathfinding (P3)	<p>While the hunter is searching for the player and hasn't found the player, the hunter should be able to generate random paths to follow around the level so that the player cannot easily predict where the hunter is going to be at any given moment.</p> <p>Furthermore, when the hunter knows the location of the player, it needs to be able to quickly calculate the shortest path to the player and travel down that path in order to make the game intense while the hunter is chasing the player.</p>	<p>When the hunter knows the players location, the hunter will need to reach that location using the shortest route therefore, I am going to implement Dijkstra's shortest route algorithm so that the hunter will be able to use this route to reach the location as quickly as possible. When the hunter knows the players location, the hunter will need to reach that location using the shortest route therefore, I am going to implement Dijkstra's shortest route algorithm so that the hunter will be able to use this route to reach the end goal</p>
AI (P2)	<p>As the hunter is able to 'hear' in my game, an AI needs to be able to decide when a sound is loud enough that it can be 'heard' by the hunter in order to engage a chase sequence.</p> <p>Also, the AI needs to be able to decide when they have seen the player, for example, if the player is in the line of sight of the hunter, then the hunter needs to be able to decide that it can see the player. However, if the player is hidden or not in direct view of the hunter then the AI needs to be able to decide that it cannot see the player and continue with its random path.</p>	<p>Firstly, as the hunter needs to be constantly checking for sounds that it can hear, I will need to use iteration. Selection will be used to check if the sound is loud enough for the hunter to hear, this will be done by also checking how many walls/objects there are between the player and the hunter and how far away the hunter is from the player in order to determine if the hunter should go to the sound.</p>
Physics (sight) (P7,	<p>As I want my game to be somewhat realistic, both the player and the hunter will have an Field Of View (FOV), although the players FOV will be different to the hunters FOV.</p> <p>Both the hunter and the player should not be able to see through walls as in real life it is impossible to see through walls with the naked eye. Furthermore, if there are objects throughout the level then the hunter should not be able to see through the object. The hunters FOV will only be in one direction which will make it more realistic, however, as</p>	<p>For the player, there will be a certain area around them that they can see, however, if this area is obstructed by objects such as walls or anything else that may be used within the game then the player will not be able to see past it. A similar thing will be implemented for the hunter, however, the hunter will only have a sector as their FOV so that they cannot see behind themselves. This is so that if the</p>

	I think it would be too unfair for the hunter to always sneak up behind the player and catch them, the players FOV will be less realistic and will be 360° so that the player has time to react to seeing the hunter.	player is behind the hunter, the hunter will not be able to see the player. However, the hunter's FOV will follow the same principle as the players, in the way that if the FOV is obstructed by a wall then the hunter will not be able to see through it
Physics (sound)	To add even more realism into my game, the hunter will be able to 'hear' the player on different occasions and therefore, there needs to be a way for the hunter AI to be able to decide if it can 'hear' the sound coming from the player. Furthermore, as sound cannot be heard through solid walls, when the sound originates from the player it should only be 'hearable' from a certain distance away from the player and it should not go through solid objects.	There will be value for the circular area of the sound that will originate from the player which will be a constant as I am using a heuristic approach in order to make sure this is not too complex. However, the sound will only be able to exist within open spaces and cannot travel through walls or other objects and, therefore, the area of effect of the sound will be the area of the circle minus the are closest objects or area blocked by walls around the player.
Win Condition	In order for the game to be a game, it needs to have an end goal which the player will have to complete, this could be something like a time limit which they have to stay hidden from the hunter for or the goal could also be finding the escape door within the level without getting caught by the hunter.	The win condition will be using selection to decide whether the player has achieved the goal in order to win the game.

Limitations:

- Graphics – As the focus of this project is the actual mechanics of playing the game and not the graphics, I cannot spend that long on RTX on 3D graphics as spending too much time in good graphics rather than the actual code itself, it may lead to code deficiencies or possible incompleteness of the program I want to create. Additionally, making the game a 3D one would mean that the calculations for things like the sound and vision of the hunter would have to be re-evaluated as there is another dimension that needs to be considered.
- Complexity – If I add too many different things to the game then not only will it oversaturate the experience for the player, but it may also affect the quality and efficiency of the code or if I could even complete the program. This means that I have to have a maximum level of complexity for my game. Furthermore, as I am using an AI for the hunter, although the AI needs to be very good, I need to think heuristically and accept a 'good enough' version of the AI which works most of the time for the tasks needed in the game.
- Multiplayer – Although I would like to make the game multiplayer so that people can play with their friends, it may take too much time and if I wanted to add true multiplayer where

people can play together on different devices then I would need to create a server that they could play on and that costs money and could also take too long.

- Accessibility – As the only language I know well enough in depth to code this program is python, accessibility may be a problem as I may not be able to make it a web game and therefore, it may be harder for people to find and play the game.

Project Requirements:

Requirements	Justification
Keyboard (R1)	This will be used for inputs while the game is running so that the player can control their character and also interact with any other aspects of the program.
Mouse (R2)	Although this may not be necessary, the user can use a mouse to navigate the menu and other menu type screens within the program
Monitor (R3)	This is needed so that the player can see the outputs of the program
Computer (system requirements): <ol style="list-style-type: none"> 1. Only need a CPU unless processor does not have integrated graphics then a GPU is also needed (R4) 2. 500 MB of RAM (R5) 3. RGB lights 4. Less than 500MB of storage space (R6) 	<ol style="list-style-type: none"> 1. As this game will not be incredibly demanding, all modern CPUs should be able to run the game easily. Furthermore, since the graphics of the game will be quite simple, a dedicated GPU will not be necessary 2. When I researched the minimum requirements of metal gear solid, its RAM requirements were 32MB which means that even half a gigabyte of RAM should be plenty for my game 3. More FPS 4. From my research Metal Gear Solid only takes up 400MB of storage space which means that 500MB should be more than enough for the game I am creating.

Success Criteria

Input:

Spec Reference	Success Criteria	Justification
i1	The player can press the 'W' button on the keyboard to move their character towards North in the level	This is so that the player can navigate in any direction through the level
i2	The player can press the 'S' button on the keyboard to move their character towards South in the level	This is so that the player can navigate in any direction through the level
i3	The player can press the 'A' button on the keyboard to move their character towards West in the level	This is so that the player can navigate in any direction through the level
i4	The player can press the 'D' button on the keyboard to move their character towards the East in the level	This is so that the player can navigate in any direction through the level
i5	The player can press the 'Shift' button on the keyboard to move their character faster (for a limited time)	This is so that the player can try and escape the hunter if the hunter is chasing the player and is also so that the player can quickly navigate through the level if need be
i6	The player can press the 'E' button in certain locations to do an action	This is so that the player can interact with any objects within the level which can help the player hide from the hunter or help the player complete the objective
i7	The mouse will be used so that the user can use the start menu to play	This is so that the user can easily navigate through the start screen as it would be harder if the user had to use the keyboard since they are probably used to using the mouse to go through menus

Process:

Spec Reference	Success Criteria	Justification
P1	On the start screen, there will be a controls button and a start button, when the controls button is pressed then the controls of the game will be displayed using an image of a keyboard	The controls screen is so that the player can quickly get to terms with the controls of the game.
P2	When the player presses start then a loading screen will appear while the	The loading screen is shown so that the game feels smoother, and the player will

	game is loaded in the background with the hunter and the player starting in different locations.	know that the game is loading. The hunter and the player start in different locations so that the hunter doesn't immediately catch the player unfairly
P3	A loop will immediately be in effect for the hunters AI so that it starts following randomly generated paths throughout the game.	This is so that the hunter is unpredictable for the player and they will not be able to learn where the hunter will move next which will create tension.
P4	There will also be another loop within the hunter's Ai that will be in effect checking whether it can see or hear the player	This is so that hunter is able to begin hunting for the player immediately through both sound and sight
P5	If the hunter hears the player then the hunter's speed should be increased (not as much as if it sees the player) and the shortest path to the location should be assigned for the hunter to follow. While the hunter is following the path to investigate the sound, its 'hearing' circle is disabled.	The speed increase is so that the hunter has the chance to catch the player if the player has not moved away quick enough, however, the hunter should still be quick enough that it can catch the player if the player takes too long to run. The hunter's hearing circle is disabled so that the player has time to escape.
P6	If the hunter sees the player then the hunter should start following the player, the speed of the hunter during the chase should be quicker than the walking speed of the player but slower than the sprint speed of the player.	This time the increase in speed is higher to make the game feel more intense, however, so that the player still has the chance to escape the hunter's chase speed needs to be slower than the players sprint speed
P7	Once the hunter starts chasing the player then the chase sequence needs to be activated which puts a red overlay on the players screen and an exclamation mark above the player's head. Furthermore, the hearing circle will also be disabled in this sequence.	The overlay is red as red is known as the colour for danger which should make the chase feel more intense and the exclamation mark above the player's head should tell the player that they have been spotted
P8	While the player is moving, there will be a circular area in which the sound can be 'heard' by the hunter, this circular area will be larger while the player is using the sprint button.	This is so that the hunter can hear the player
P9	The hunter will also have a radius around them in which they can hear the player, this circle changes depending on the distance and number of walls between the player and the hunter.	This gives the hunter an area in which they have the possibility to hear the player
P10	If the hunter and the player's circles overlap, then depending on the amount of overlap the hunter will have a	This gives the user a feeling that the hunter is deciding when it has heard a suspicious sound in the game to investigate and adds some realism.

	different chance on whether they can hear the player or not.	
P11	If the player is out of the hunters FOV during the chase sequence, then the hunter will follow a random path at a lower movement speed in order to search for the player before reverting to the unalerted version after a few seconds	This makes it more intense as the hunter does not immediately back off as soon as the player is out of sight and makes the game more intense as the player may have to spend a little more time running.
P12	If the hunter catches the player then the game will be ended, showing the end screen (O6) and the user will be returned to the start screen.	This so that the player can be flawlessly returned to the beginning screen to restart the game.
P13	When the player presses the use button ('E') while next to a hiding spot then the players character will be non-existent to the hunter while they are still hiding	This is so that instead of running, the player can also have this option to escape the hunter
P14	If the player is within the hunter's FOV and player tries to hide using the hiding spot the hunter will still be chasing the player and will catch them if the player is still in that same hiding spot	This makes the game feel more realistic because if you see someone hiding in front of you, you will already know that the person is in that place.
P15	When the player presses the use button 'E' while next to a lever then it will change the light above the lever from red to green and update the objective by one.	As the objective of the game will be to activate a certain number of levers/switches in order to escape, the player's objective needs to be updated each time the player activates a lever so that they know how many more are left in the level.
P16	When the player activates the lever then the hunter is alerted to that location after a short amount of time of a few seconds or less.	This will make the game more tense for the player as each time they activate a switch or lever, then they know that the hunter will be coming to them.
P17	When the player completes the objective (activating all the levers/switches) then the exit door opens, and text is shown which says something like 'The escape door has opened!' and objective completion displayed in the corner will change to escape	This is so that the player knows what they need to do next which is find the escape within the level so that they can complete the game.
P18	If the player moves through the escape door then a screen should be displayed saying 'You have escaped' and it also shows how long it took them to complete the game just under the text.	The text is so that the player knows they have completed the game and the time is so that the player knows how quickly they accomplished the game. Furthermore, the time taken to beat the level will make it so that the player wants to play the game

		again and again to see how much faster they can complete the game
P19	When the start button is pressed, it takes the user to a different screen which shows different levels of difficulty which will be easy (3 levers); normal (5 levers); hard (7 levers); insane (9 levers) and nightmare (13 levers) - depending on how big the level will be, some of the difficulties may not be included	This is so that at the start of the game the player can get used to how the game functions and what they need to do to win. The other difficulties will keep the player interested in the game for longer, as after one difficulty, they have to attempt to beat the next difficulty
P20	The FOV of the hunter will be a sector of a circle which originates from the hunter and the chord will be the length between the walls. If the hunter is facing a wall then the FOV of the hunter will only reach up until that wall.	This is so that the hunter has a realistic field of view just like people would in real life and to make sure that the hunter cannot see through the walls which can cause a bug, the hunter cannot see past the wall.
P21	The FOV of the player will be anything that isn't behind a wall so if a player is standing in the middle of a long hall then they should be able to see to either end of the hall but not beyond the walls next to them or around the corners at the end of the hall.	Although this may not be as realistic as the player is also able to see behind them at all times, the game would be too hard if the player wasn't able to see behind them as the hunter would be able to constantly sneak up on the player from behind without the player knowing.
P22	The hunter will only be able to catch the player if the hunter is in extremely close proximity to the player	This is so that the hunter is not able to catch the player from the other side of the level as that would be unfair
P23	When the player presses shift while moving, then the player should be moving at twice the speed they were moving at originally which will be slightly quicker than the hunter's chase speed.	This adds another way for the player to escape from the hunter and the speed will be higher than the hunter's chase speed so that the player can escape from the hunter.
P24	As the player presses shift (while moving) their stamina bar at the bottom right of the screen will decrease at the same rate as the value for stamina the player has left.	This is so that the player knows how much stamina they have left so they can strategically use it for sprinting when needed.
P25	The players stamina will only replenish if the player is not sprinting (pressing the shift button) and when the stamina value is being replenished then the stamina bar will refill at the same rate.	This makes the game more realistic as the player has limited stamina just like in real life.
P26	The level itself will have a simplistic maze design, resembling the inside of some sort of building. However, there will be no dead ends within the level.	It will resemble the inside of a building so that it feels more realistic and it has a maze design so that it can make the game more challenging. However, there must be no dead ends so that the player can

		escape the hunter as a dead end would make the player get stuck with no escape.
--	--	---

Output:

Spec Reference	Success Criteria	Justification
O1	The player's sprite moves upwards if the player presses 'W'	'W' is widely regarded as a forward or North moving motion in games and in my game upwards is the forward moving motion which should make it easier for the player to learn the controls of the game
O2	The player's sprite moves downwards if the player presses 'S'	'S' is widely regarded as a backwards or South moving motion in games and in my game downward is that motion which should make it easier for the player to learn the controls of the game
O3	The player's sprite moves left if the player presses 'A'	'A' is widely regarded as a left or West moving motion in games and in my game left is that motion which should make it easier for the player to learn the controls of the game
O4	The player's sprite moves right if the player presses 'D'	'D' is widely regarded as a right or East moving motion in games and in my game right is that motion which should make it easier for the player to learn the controls of the game
O5	If the player tries to move in a direction where an object is directly in front of them, the player's sprite should not go through it	This is so that the game is more realistic so that it is more immersive for the player.
O6	If the player gets caught by the hunter then a 'You Have Been Caught' screen will be displayed and the player will be returned to the main menu.	This screen is displayed to inform the player that they have lost the game and as they are returned to the playing screen straightaway, it is easier for the user to play again.
O7	If the player presses 'E' while they are next to something that can be used e.g. a hiding space then the player's sprite should act accordingly and text in relation to the act will be shown e.g. 'Hidden'.	This is so that the player is informed that they are carrying out an action as sometimes even if the player presses the relevant button they might not be carrying out the action in the game. This makes it easier for the player to know whether their character is doing as intended.

O8	When the player presses the 'shift' button while moving in a direction then the player's sprite should appear to be moving quicker on the monitor.	This is similar to O7 in the way that it is so that the player knows that they are sprinting at the moment as it looks different
O9	If the player complete the objective then a message should appear displaying 'You have escaped' or something similar and return the player to the start page	This is so that the player knows that they have beaten the game and as they are returned straight to the starting menu, the user can play the game again.
O10	Depending on the objective of the player a relevant display in the corner of how close they are to completing the objective should be in the corner of the screen.	This is so that the player knows how much longer they need to evade the hunter for and as the objective gets closer to being finished it also makes the game more intense for the player.
O11	While in the game, the player should have a limited field of view (Bigger than the hunter's FOV), however, this is 360 degrees around the player.	The players FOV is bigger than the hunter's so that the player has time to escape when they see the hunter. The player will have a 360 FOV so that the hunter cannot sneak up behind the player and unfairly make the player lose
O12	If the player has no more sprinting energy left and the player presses shift then text will be displayed near the top of the screen saying 'Wait for your energy to replenish'	Again, similar to O7 and O8, this is to inform the player that they have run out of energy and need to wait (not sprint) in order to replenish their energy to sprint again.

Aesthetic:

Spec Reference	Success Criteria	Justification
Ae1	If the hunter has seen the player and is chasing them, then an alarm effect or something similar should be added on top of the players screen	This is similar to some of the outputs (O7,O8,O12) in the aspect that they are so that the player will clearly know that the hunter is chasing them
Ae2	The level itself will be quite basic, with the walls being rectangles while will block the player into a sort of maze which resembles a structure	As the game is meant to be realistic, the walls will be formed together in such a way that the player may feel like they are inside a building which will make the game more immersive and intense as they need to escape out of this unknown building
Ae3	The player's sprite will look simple with a big oval for the body, a sphere for the head and 2 longer ovals for the arms of the sprite	Since this game is not made for graphics, the sprite does not need to be too complex as it will take up too much time and be inefficient
Ae4	The hunter's sprite will be similar to the players except the hunter will have a different skin to the player	This is again (like Ae3) to save time as it would take too long to make the hunter's sprite look completely different

Ae5	On the players screen places where the player can interact with will look different to the surrounding environment	This is so that the player can easily recognise that this is a place where they are able to do something e.g. hiding
-----	--	--

Design

Decomposing the problem

I have decomposed my main game into 6 key parts which are:

- 4 screens:
 - This is interface that will act like a menu for the user to go through
- Hunter:
 - This is the hunter AI which contains its pathfinding, hearing and decisions
- Player:
 - This is the player and the different calculations and things that the player needs to be able to do while in the game
- Map:
 - This is what the hunter and the player will be travelling through and it needs to be a grid with objects
- Physics:
 - Contains most of the realism calculations and the laws of the game such as collisions and projectiles
- Round Ends:
 - These are the conditions for the round to end

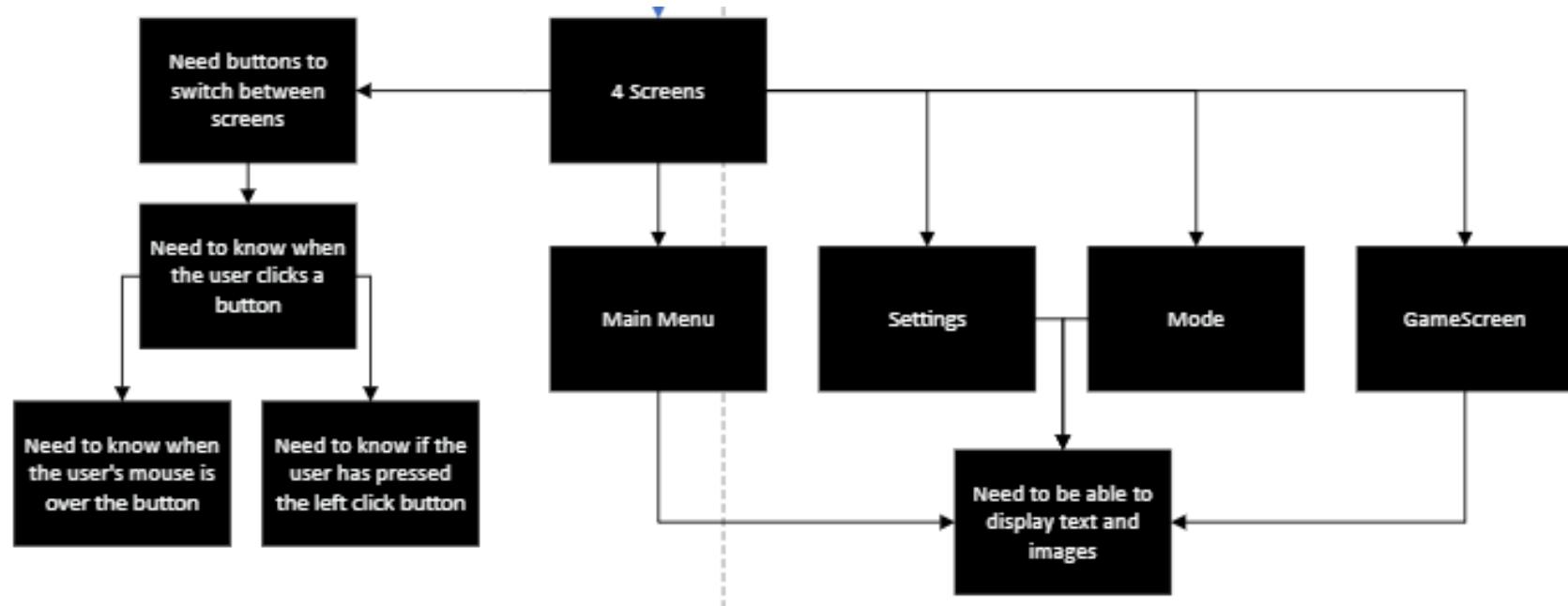
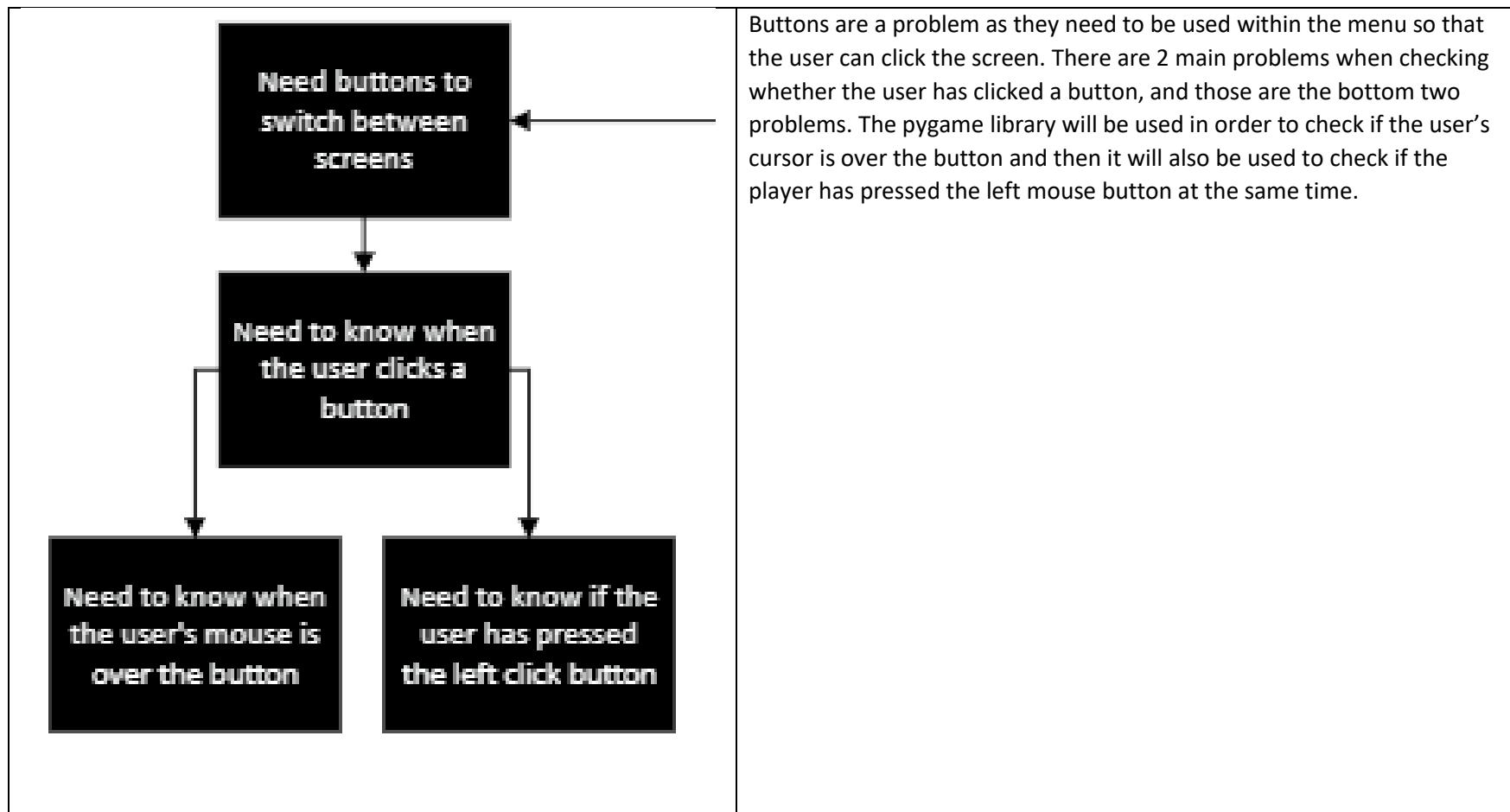
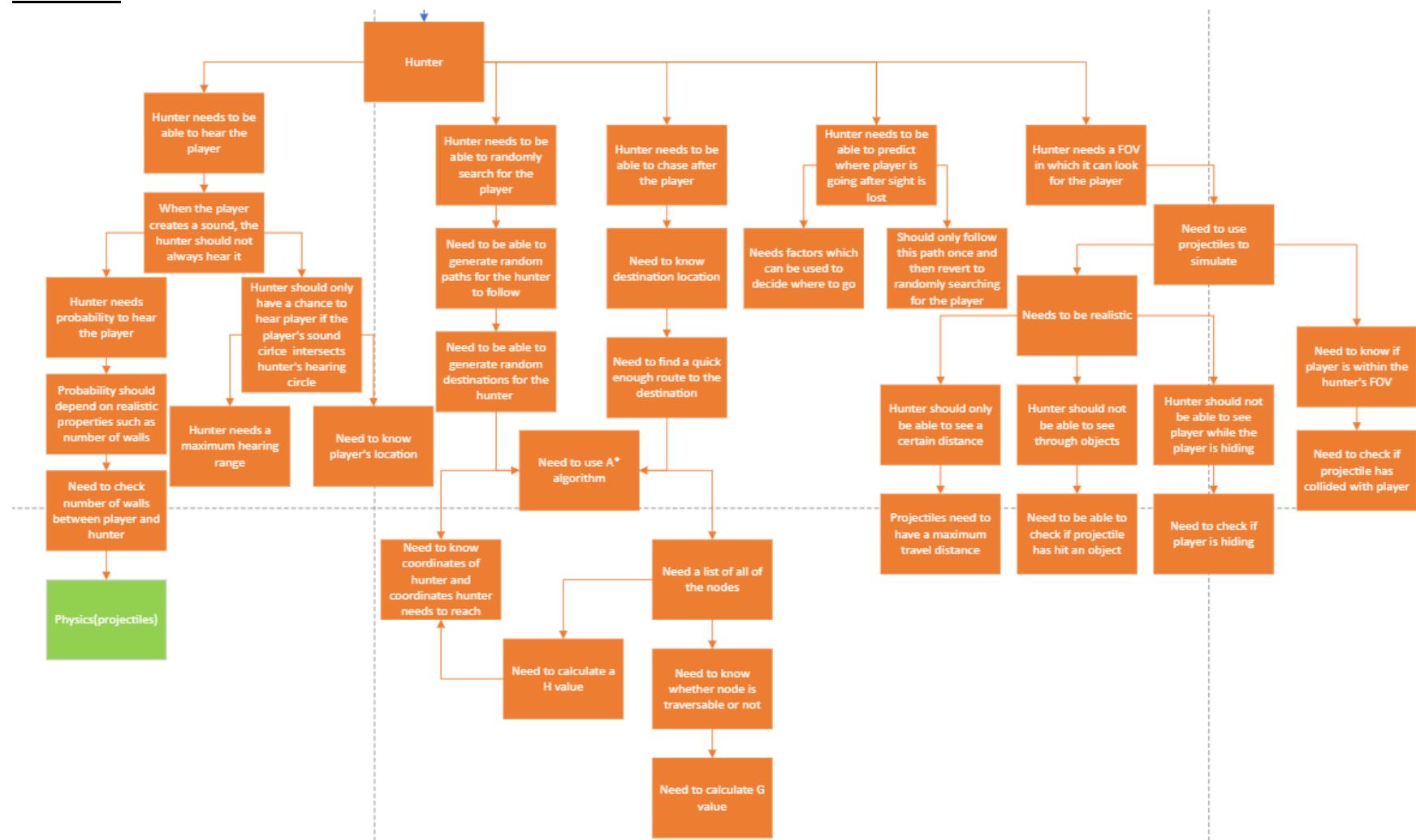
4 Screens:

Diagram	Justification
<pre> graph TD A[Main Menu] --> B[Need to be able to display text and images] C[Settings] --> B D[Mode] --> B E[GameScreen] --> B </pre>	<p>These are problems as I need to make sure that the user is able to easily navigate through the menu and also is able to see things on the screen. The first 3 screens just need to display static images, text and have some buttons which transfer the user to a different screen. However, the GameScreen needs to also be able to update these images on the screen as the player and hunter move or interact.</p>



Hunter:

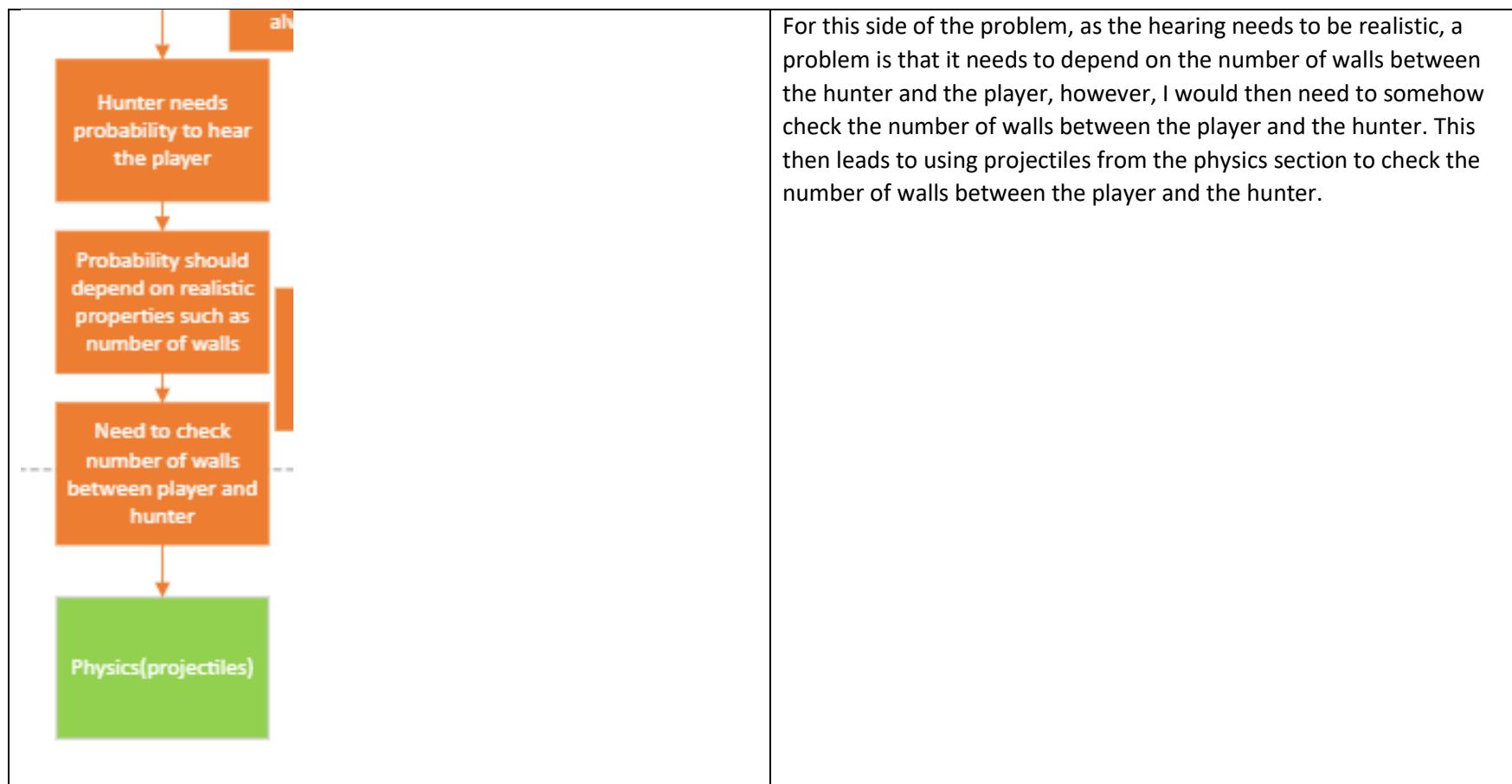
Diagram

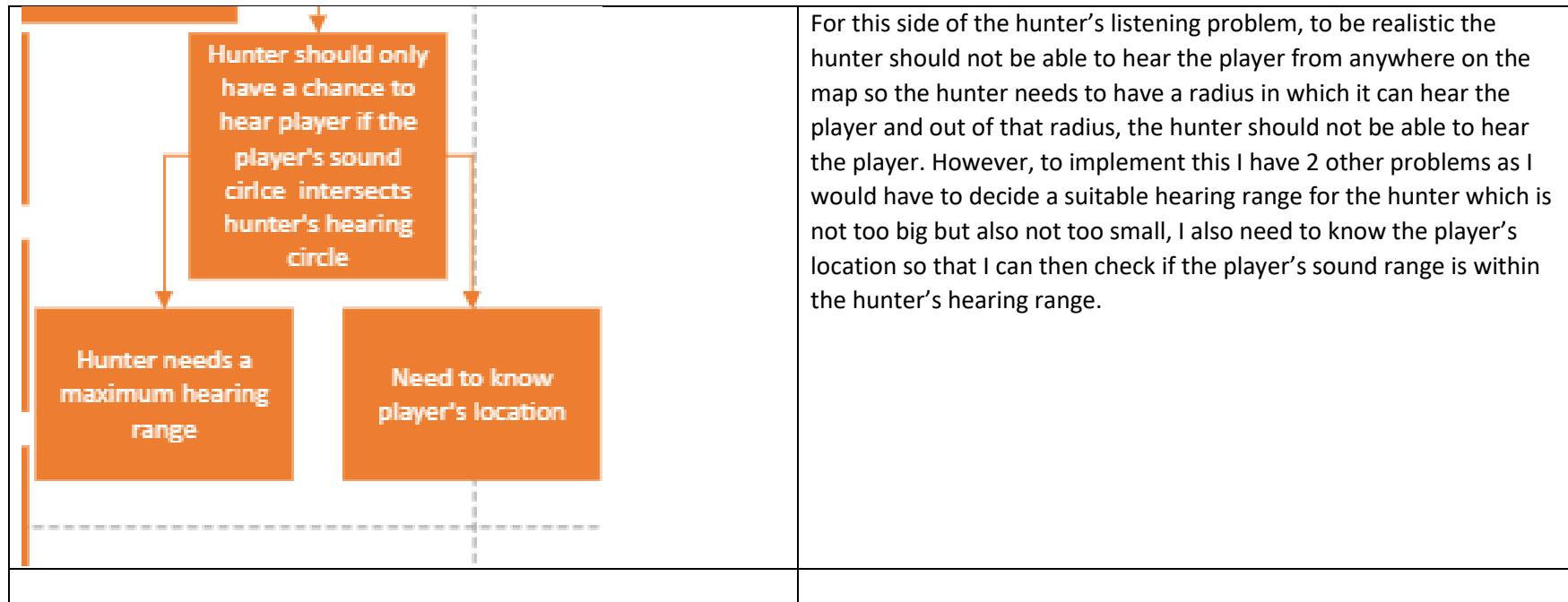
Justification

<pre> graph TD Hunter[Hunter] --> Hear[Hunter needs to be able to hear the player] Hunter --> Search[Hunter needs to be able to randomly search for the player] Hunter --> Chase[Hunter needs to be able to chase after the player] Hunter --> Predict[Hunter needs to be able to predict where player is going and sight is lost] Hunter --> FOV[Hunter needs a FOV in which it can look for the player] </pre>	<p>In this section the main problem is coding the hunter's AI which I have broken down into 5 main parts. I did this because these are the main problems with the hunter's AI, the first one is the hunter's hearing abilities, the next 3 are to do with the hunter's pathfinding abilities, although they are slightly different from one another, and the last one is to do with the hunter's seeing abilities</p>
<pre> graph TD Hear[Hunter needs to be able to hear the player] --> NotAlways[When the player creates a sound, the hunter should not always hear it] NotAlways --> Probability[Hunter needs probability to hear the player] NotAlways --> CircleIntersection[Hunter should only have a chance to hear player if the player's sound circle intersects hunter's hearing circle] </pre>	<p>For the hunter's hearing abilities, I broke it down into 2 more branches as one side of the problem is how the hunter's chance of hearing the player should vary accordingly to the surroundings and the other is how the hunter's chance of hearing should vary according to the player's position.</p>

Name: Muqtasid Zayyan Dar

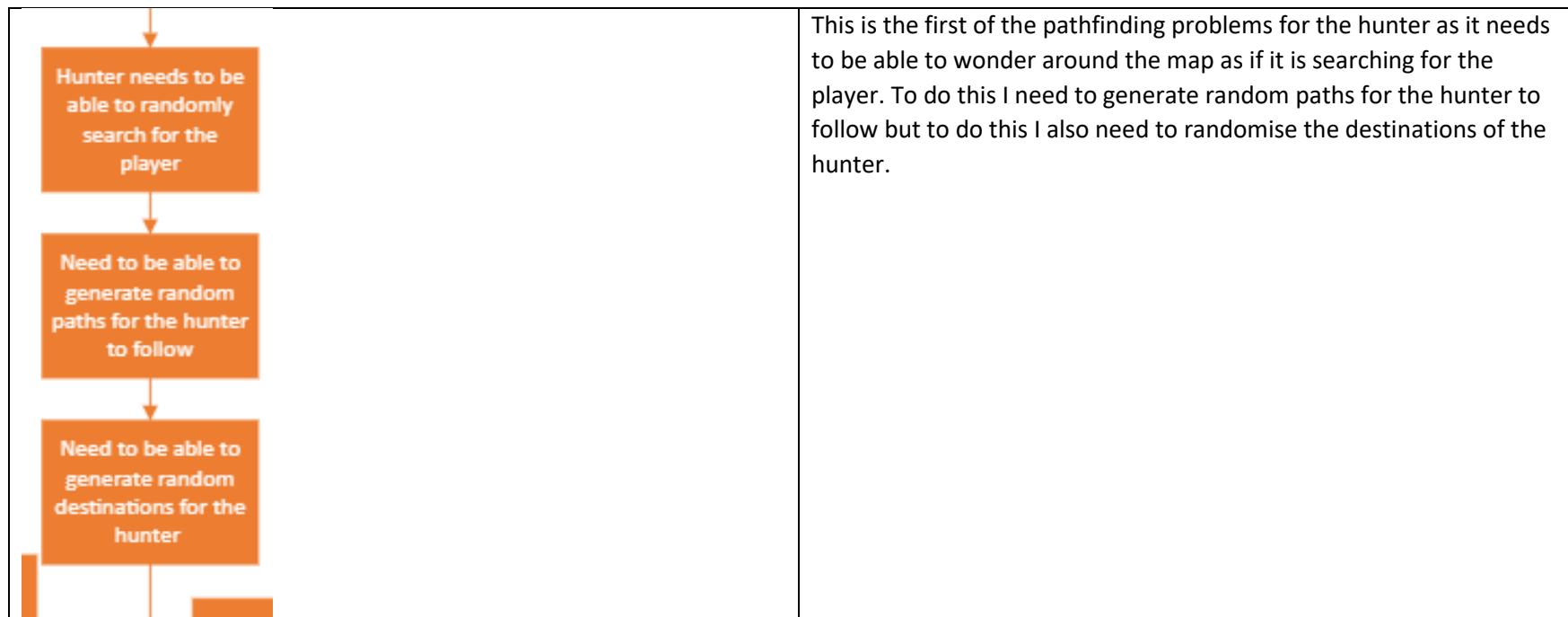
Project title: Manhunt





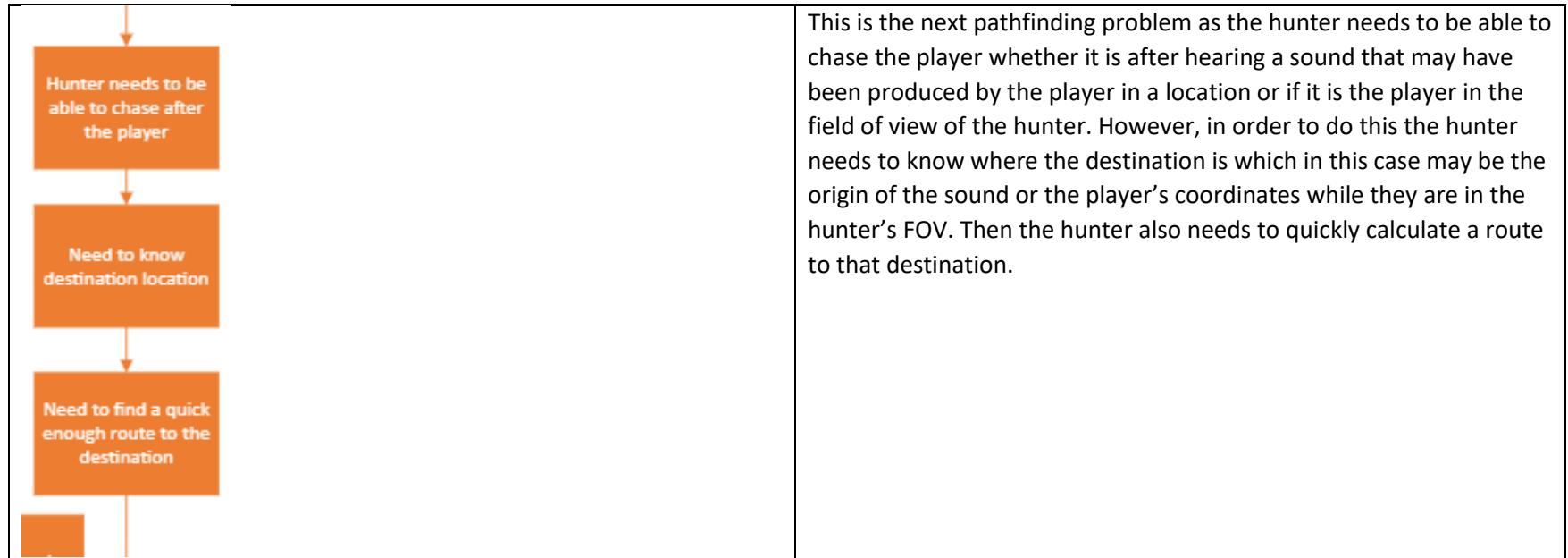
Name: Muqtasid Zayyan Dar

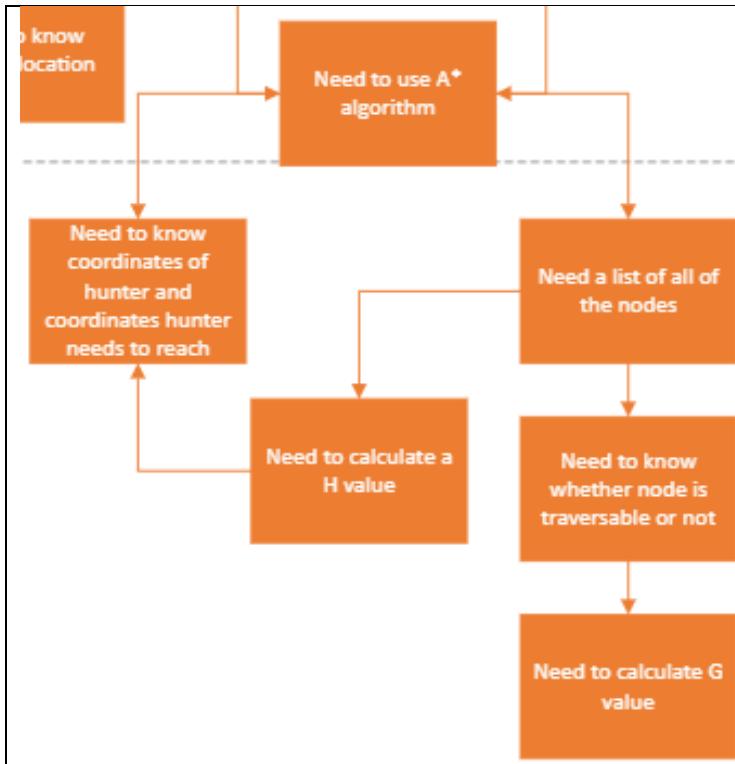
Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



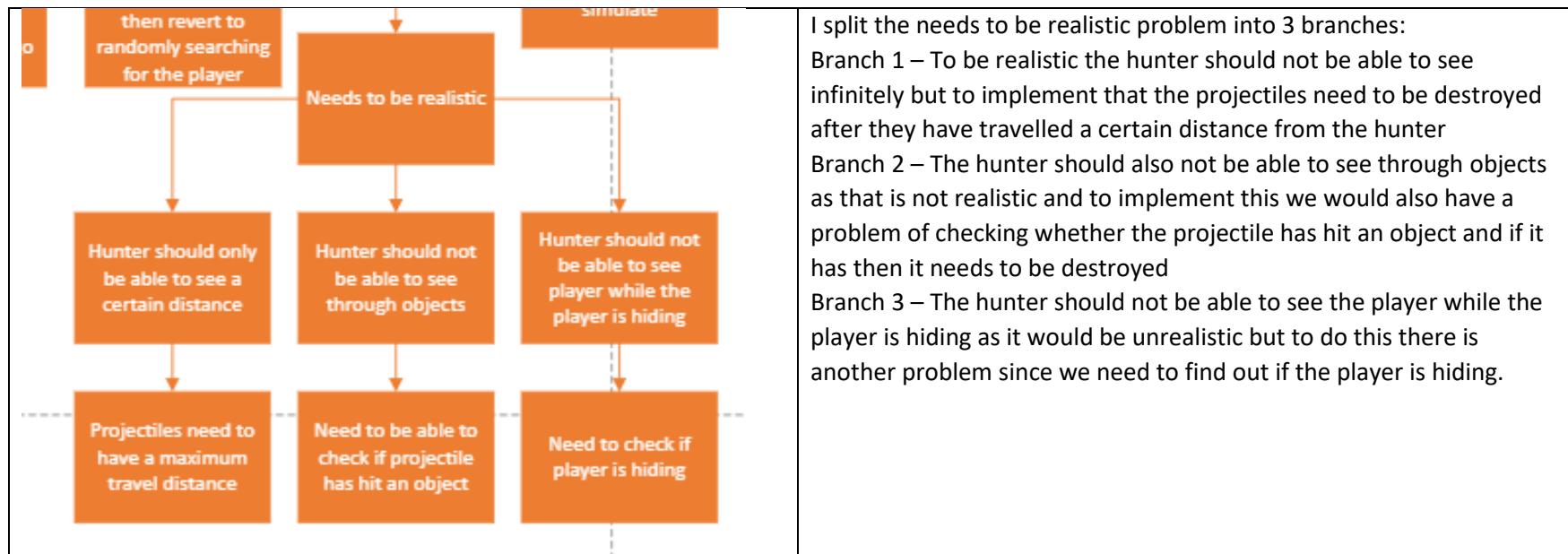


In both of these situations, the next problem is the pathfinding algorithm itself, which in this case is the A* algorithm as I believe it will find a quick path in the shortest time. However, to use the A* algorithm there are some preconditions which form some of the other problems the branch for there. Firstly, I need a list of all of the nodes in the map, and then I also need to know which nodes are traversable (the floors) and which aren't (the walls) as the hunter can walk on floors but not walls. A heuristic value for the distance from the hunter and the destination needs to be calculated however, the coordinates of the destination and the coordinates of the hunter need to be known before that can happen. After all of this the G value also needs to be calculated to be used in the F cost in the algorithm

<pre> graph TD A["Hunter needs to be able to predict where player is going after sight is lost"] --> B["Needs factors which can be used to decide where to go"] A --> C["Should only follow this path once and then revert to randomly searching for the player"] </pre>	<p>This is the last pathfinding situation, where the hunter needs to be able to predict where the player has gone after the hunter has lost sight of the player. This can be broken down into 2 smaller problems, the factors affecting where the hunter should decide to go and how many times the hunter should try and predict where the player has gone.</p>
<pre> graph TD A["Hunter needs a FOV in which it can look for the player"] --> B["Need to use projectiles to simulate"] B --> C["Needs to be realistic"] C --> D["Hunter should not"] B --> E["Need to know if player is within the hunter's FOV"] E --> F["Hunter should not"] </pre>	<p>For the FOV the main problem is that the hunter needs to be able to search for the player. This problem can be broken down into how it should be simulated which I decided to use projectiles, however, this also splits into 2 more branches, as the FOV needs to be realistic and the hunter needs to be able to detect the player if the player is within the hunter's FOV</p>

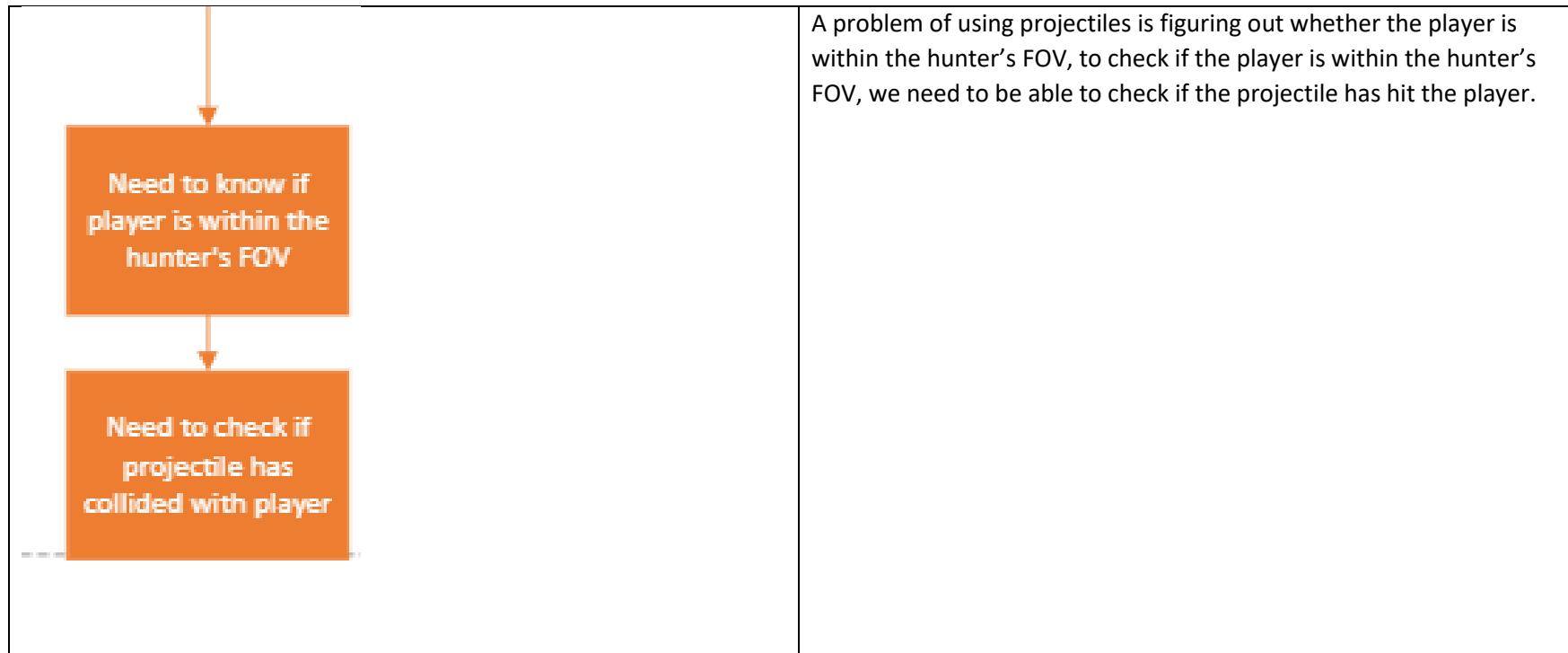
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



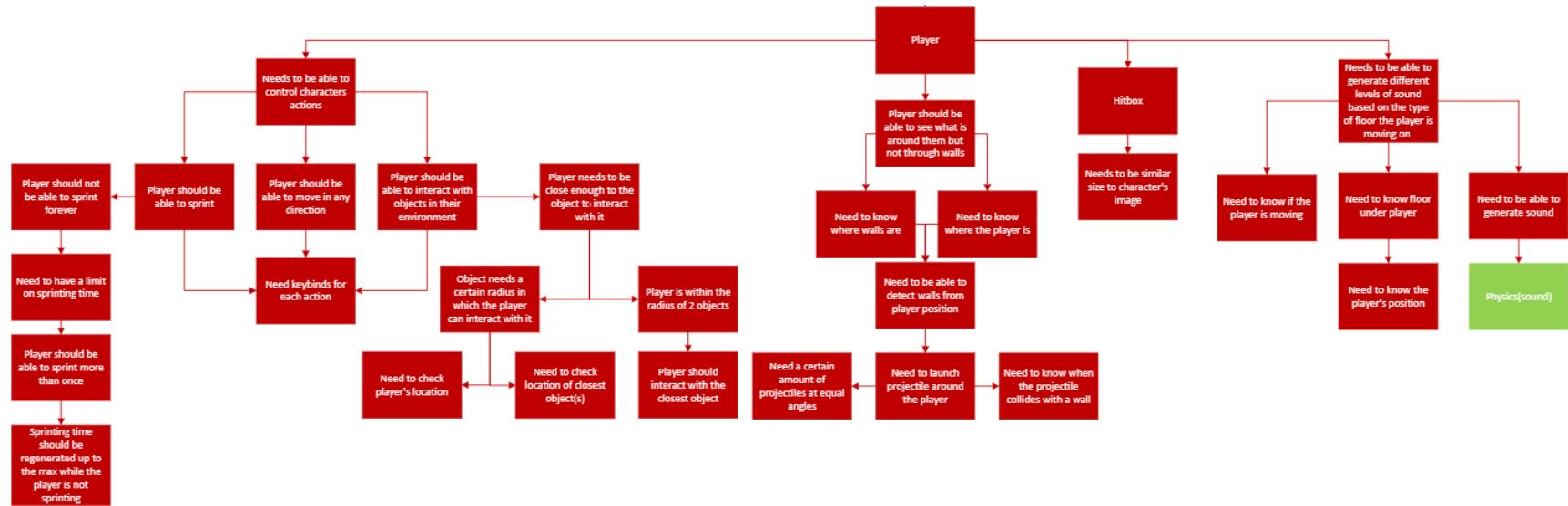
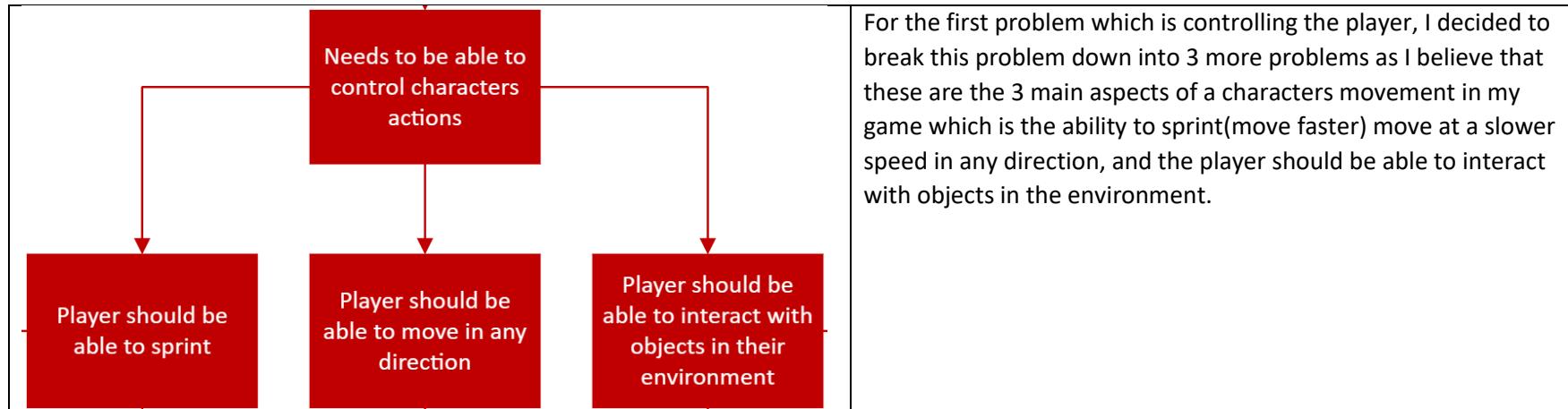
Player:

Diagram	Justification
<pre> graph TD Hitbox[Hitbox] --- Sound[Needs to be able to generate different levels of sound based on the type of floor the player is moving on] Sound --- Position[Need to know the player's position] Position --- Floor[Need to know floor under player] Floor --- Moving[Need to know if the player is moving] Moving --- Image[Need to be similar size to character's image] Image --- Hitbox </pre> <p>This simplified diagram shows the relationships between the Hitbox, Player visibility, and Player actions components from the main flowchart.</p>	<p>I decided to split the player into 4 main problems, which are the actions the player can carry out, what the player should be able to see, what the hitbox of the player should be and how the sound should be generated from the player. I chose these as I believe these are the 4 main aspects of the player in the game</p>

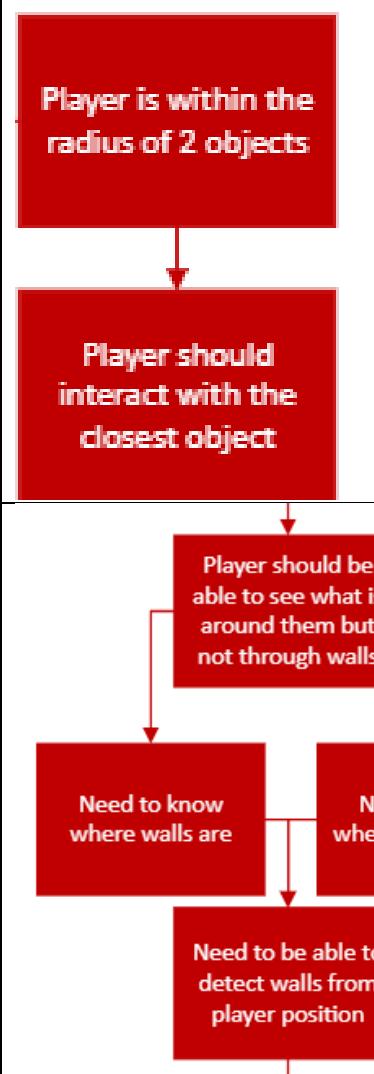
Name: Muqtasid Zayyan Dar

Project title: Manhunt



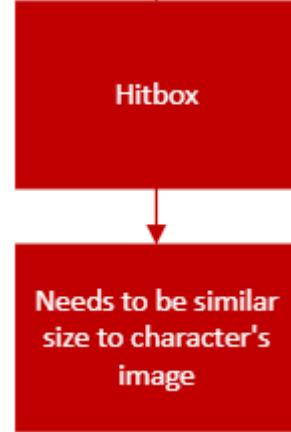
<pre> graph TD A[Player should be able to sprint] --> B[Player should not be able to sprint forever] B --> C[Need to have a limit on sprinting time] C --> D[Player should be able to sprint more than once] D --> E[Sprinting time should be regenerated up to the max while the player is not sprinting] </pre>	<p>I broke down the sprinting problem into a series of other problems. The first problem is because I want my game to be realistic, therefore, the player should not be able to sprint forever. A problem for this is that there then needs to be a time limit for how long the player can continuously sprint so that they are not able to sprint forever. However, I also need to make sure that even after the player used up all of their sprinting time that the player is able to do it again as the player may need to escape the hunter more than once. Finally, this leads to the final problem of replenishing the sprinting time while the player is not sprinting so that the player can then sprint again.</p>
<pre> graph TD A[Player should be able to sprint] --> E[Need keybinds for each action] B[Player should be able to move in any direction] --> E C[Player should be able to interact with objects in their environment] --> E D[Object certain which can interact with the character] --> E </pre>	<p>For the three branches from controlling the character's actions, the main problem for all of these is that the player needs to be able to control their character using the keyboard and the relevant keys, therefore, there needs to be keys which link to a certain action.</p>

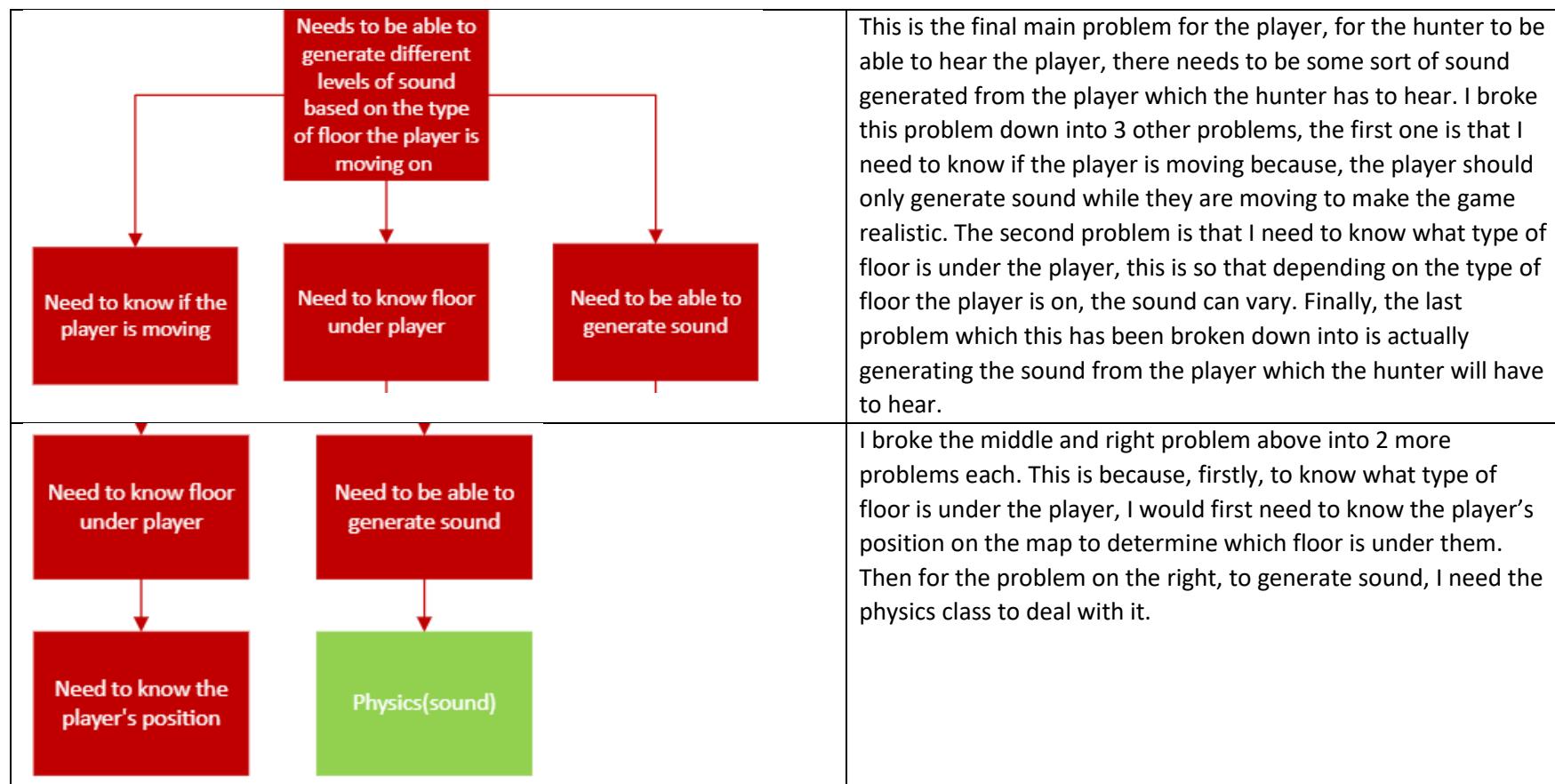
<pre> graph LR A["Player should be able to interact with objects in their environment"] --> B["Player needs to be close enough to the object to interact with it"] </pre>	<p>A problem with the player being able to interact with objects in their environment is that I want the game to be realistic so the player must be close enough to the objects to be able to interact with them</p>
<pre> graph TD A["Player should be able to interact with objects in their environment"] --> B["Player needs to be close enough to the object to interact with it"] B --> C["Object needs a certain radius in which the player can interact with it"] C --> D["Player is within the radius of 2 objects"] </pre>	<p>I split this problem down into 2 more problems as the object needs a certain radius in which the player is able to interact with the object so that I can then check whether the player is within the radius of the object in order to be able to interact with it. Another problem could be that the player is close enough to interact with 2 different objects at the same time if the radii of the 2 objects overlap</p>
<pre> graph TD A["Player should be able to interact with objects in their environment"] --> B["Player needs to be close enough to the object to interact with it"] B --> C["Object needs a certain radius in which the player can interact with it"] C --> D["Need to check player's location"] C --> E["Need to check location of closest object(s)"] </pre>	<p>I split this problem down into these 2 problems because when the player wants to interact with an object then we first need the player's location and then the location of the closest objects, then these 2 can be compared in order to determine whether or not the player can interact with an object nearby.</p>

 <pre> graph TD A[Player is within the radius of 2 objects] --> B[Player should interact with the closest object] B --> C[Player should be able to see what is around them but not through walls] C --> D[Need to know where walls are] C --> E[Need to know where the player is] D --> F[Need to be able to detect walls from player position] </pre>	<p>A problem when the player is within the radius of 2 different objects is that the player may try and interact with 1 and instead interact with 2 and break the game as their areas overlap, therefore, to resolve this the player should only interact with the closest object to them to ensure that this does not occur. Furthermore, when designing the map, this will be taken into account to ensure that this cannot occur.</p>
	<p>For this main problem of the player, I decided to first split it into 2 different problems as for the player to be able to see up till the walls but not through them, the position of the walls relative to the player needs to be known so because of this, the position of the player and the walls need to be known. Then these 2 problems are broken down into another problem because after knowing the position of the walls, you would need to know the position of the player relative to the walls to be able to figure out what the player should be able to see.</p>

Name: Muqtasid Zayyan Dar

Project title: Manhunt

 <p>Need a certain amount of projectiles at equal angles</p> <p>Need to launch projectile around the player</p> <p>Need to know when the projectile collides with a wall</p>	<p>I broke down the problem above into another problem which leads to 2 more problems. Firstly, A problem is that since the player needs 360 vision, projectiles need to be launched from the player in all directions in order to simulate 360 vision, the projectiles also allow me to figure out the position of the player relative to the walls, as they originate from the player. This problem is then broken down into further problems as for the projectiles to be launched from the players in all different directions, they need to be launched at different angles from the player. I also broke the problem down to the other problem on the right as I need to be able to detect when the projectile collides with a wall in order to figure out up till what point the player can see</p>
 <p>Hitbox</p> <p>Needs to be similar size to character's image</p>	<p>This is the next main problem which is the hitbox of the player, the player needs a hitbox so that I can detect when the player collides into objects such as walls or the hunter. A problem for the hitbox is that it needs to be a similar size to the characters image so that it seems like the image of the player is the actual character to the player.</p>



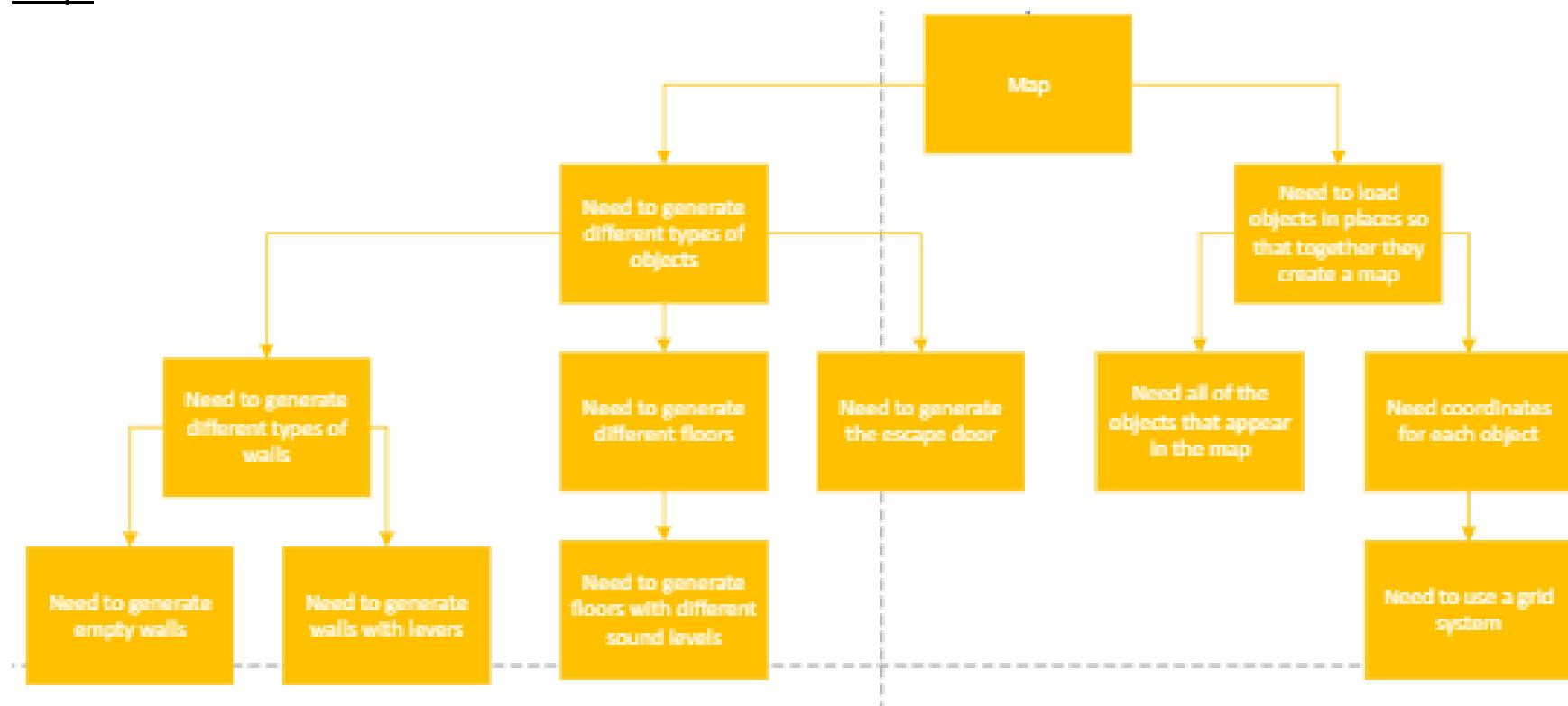
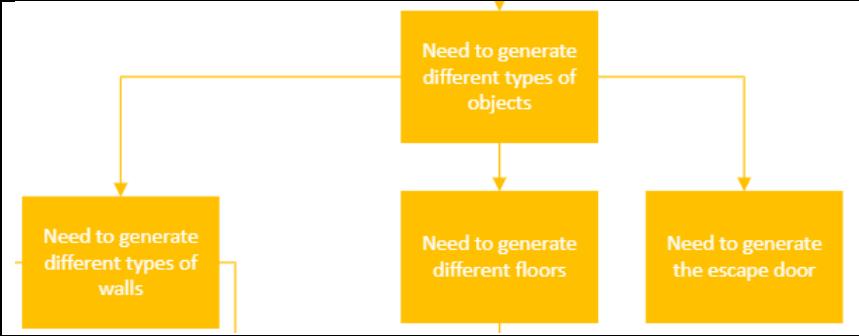
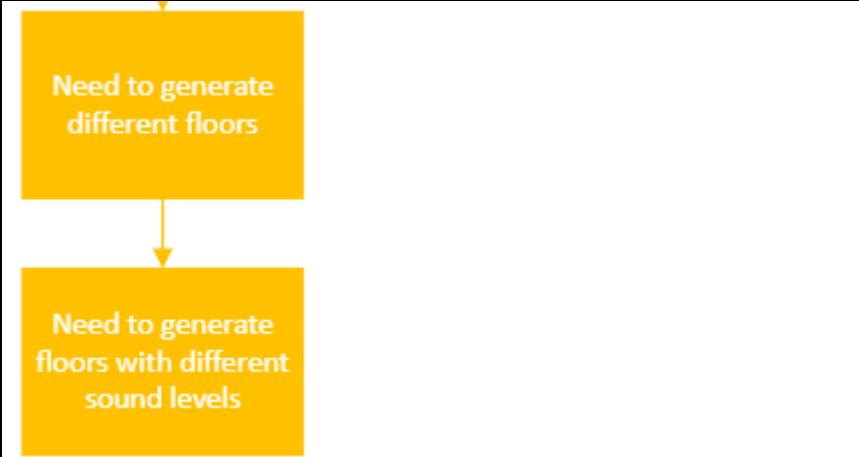
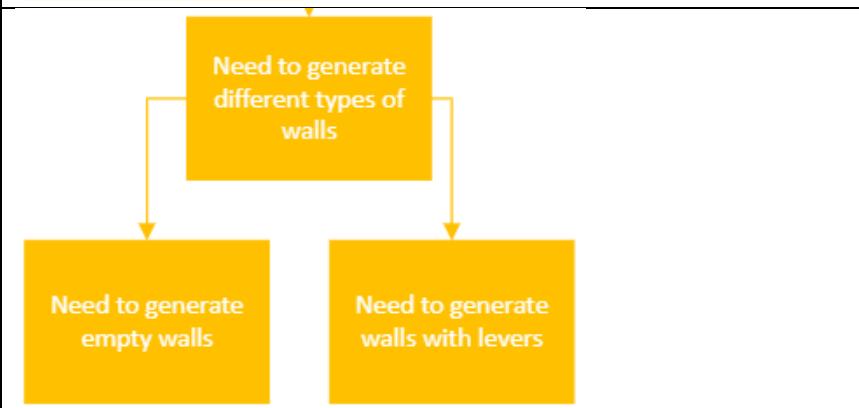
Map:

Diagram	Justification
<pre> graph TD Map[Map] --> Obj[Need to generate different types of objects] Map --> Load[Need to load objects in places so that together they create a map] </pre>	I decided to split the main problem into 2 different problems, as the map needs to generate the objects to be able to put them in the map and the map needs to be able to load these objects into the correct place in the map.

 <pre> graph TD A[Need to generate different types of objects] --> B[Need to generate different types of walls] A --> C[Need to generate different floors] A --> D[Need to generate the escape door] </pre>	<p>The problem above on the left is broken down into 3 more problems as each different type of object in the map needs to be generated for the map. To solve the problem above, I would have to figure out how to generate all of the objects below it for the map (the walls, floors, and door)</p>
 <pre> graph TD A[Need to generate different floors] --> B[Need to generate floors with different sound levels] </pre>	<p>As there are going to be different types of floors in the map, different types of floors need to be generated for the map.</p>
 <pre> graph TD A[Need to generate different types of walls] --> B[Need to generate empty walls] A --> C[Need to generate walls with levers] </pre>	<p>Similar to the floors, there are multiple types of walls, therefore, I decided to break down the generating different types of walls problem into 2 more problems, 1 for each type of wall as I would first have to figure out how to generate empty walls for the map and then generate walls with levers in the map.</p>

<p>Need to load objects in places so that together they create a map</p> <p>Need all of the objects that appear in the map</p> <p>Need coordinates for each object</p>	<p>For the problem of loading the objects together to create a map, I have broken this down into 2 problems, this is because I would need all of the objects which would need to appear in the map and then I would also need coordinates for each object in order to be able to place them in a specific place in the map.</p>
<p>Need coordinates for each object</p> <p>Need to use a grid system</p>	<p>To make coordinates for each object, I would first need some sort of grid system which uses coordinates like a graph in order to have specific places for each object.</p>

Physics:

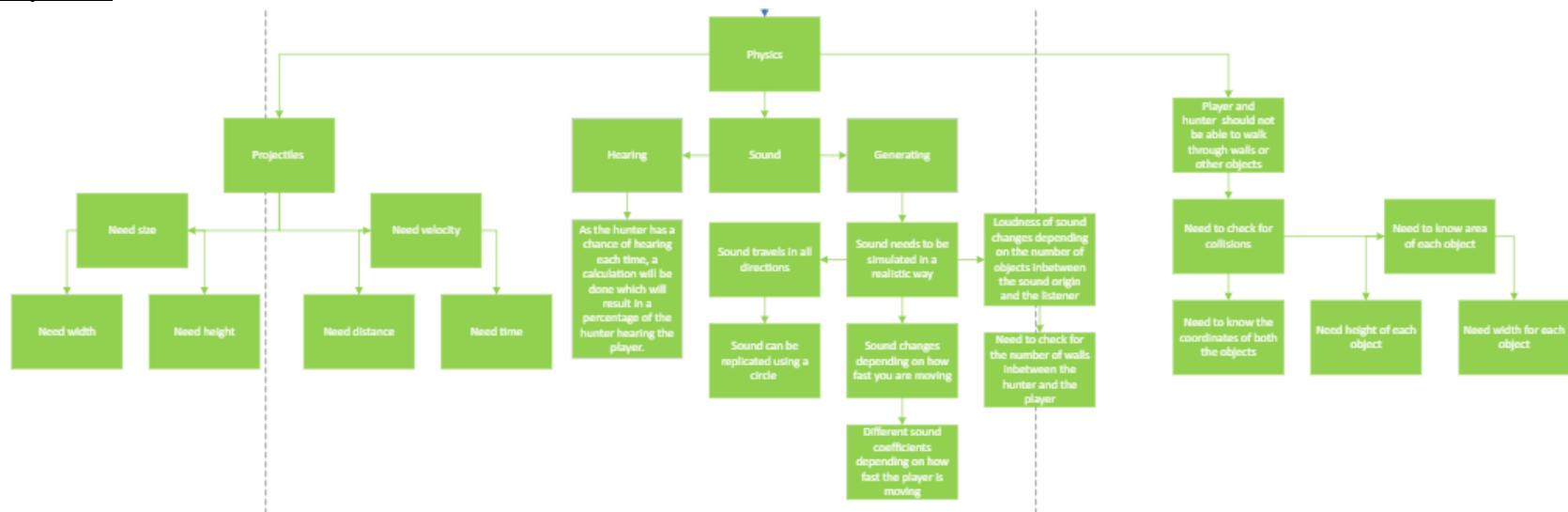
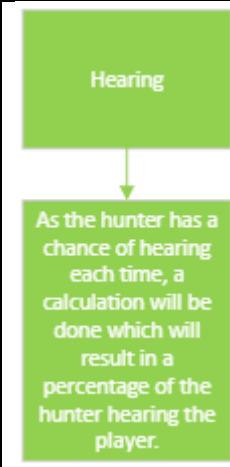
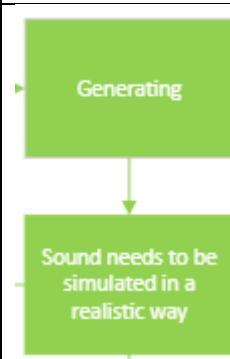
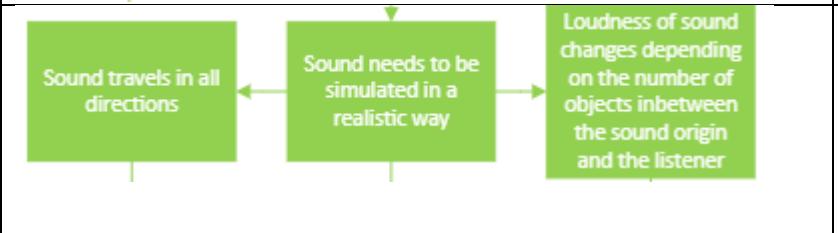


Diagram	Justification
<pre> graph TD Physics[Physics] --> Projectiles[Projectiles] Physics --> Sound[Sound] Physics --> Collision[Player and hunter should not be able to walk through walls or other objects] </pre>	<p>I broke the main problem of physics into 3 different problems:</p> <p>Sound – as this needs to be a somewhat realistic component of the game so it needs to be similar to real life.</p> <p>Projectiles – This will need to be used for the FOV of the hunter and the player and also some parts of the sound to detect things when the projectiles collide with things like walls or the player. Therefore, this needs to use physics in order to work properly</p> <p>Collisions – Although the last box does not say collisions in it, it is essentially what I am eluding to by saying that the hunter and player should not be able to walk through objects as the game has to be realistic</p>

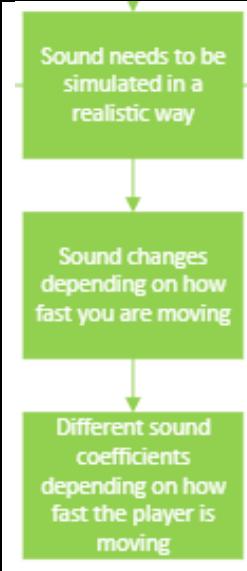
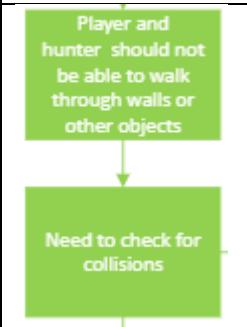
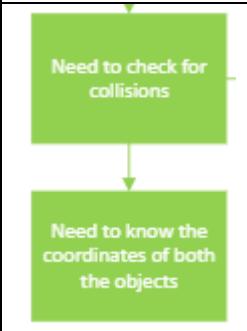
Name: Muqtasid Zayyan Dar

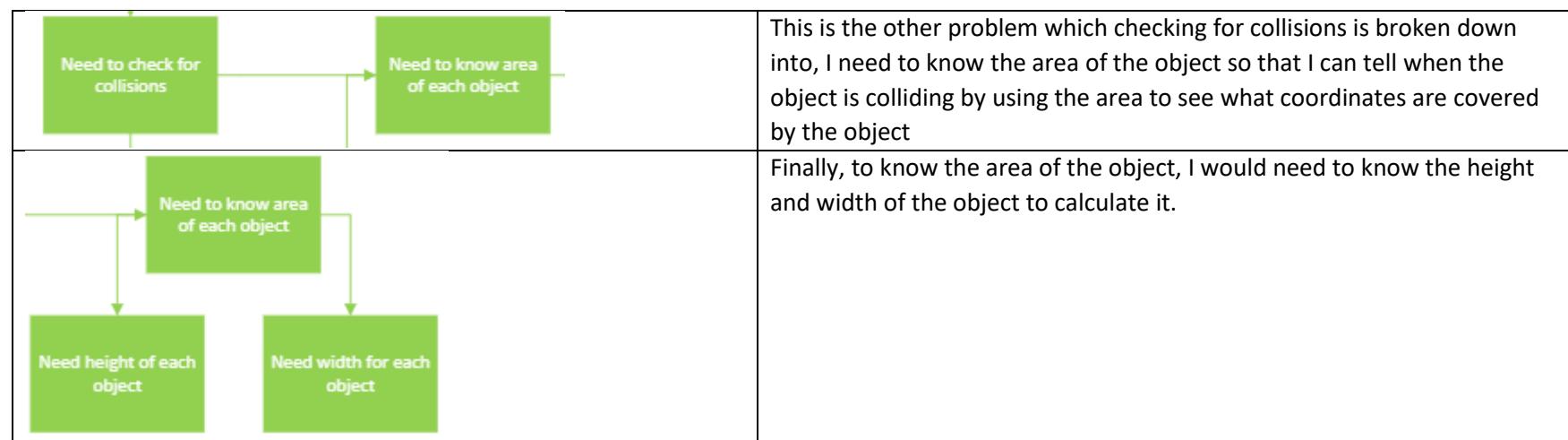
Project title: Manhunt

<pre>graph TD; A[Need size] --> B[Need width]; A --> C[Need height]</pre>	I broke the projectile problem into 2 other problems, this is the first one as the projectiles need a specific size in order to detect objects when they collide into them. This is because if the projectiles are too small then I may need a lot more of them for the FOV of the player or the hunter and if they are too big, then they may not accurately detect what they collide with as they may collide with many objects at once. This problem can be broken down further into the width and height for the projectiles as to have a size, they need each of these properties.
<pre>graph TD; A[Need velocity] --> B[Need distance]; A --> C[Need time]</pre>	This is the second problem is the speed of the projectiles as I need to make sure that they are fast so that the FOV for the hunter and player can be updated quickly but not too fast that with every update of the screen they miss objects as they have been moved past them. Then this problem can be broken down further into the distance and time as these are both needed for an object to have a velocity.
<pre>graph TD; A[Hearing] <--> B[Sound]; B <--> C[Generating]</pre>	For the second main problem of physics, I broke it down into 2 main components as there are 2 ways to interact with sound in real life, both of which are needed in my program: Hearing sound and generating sound.

 <p>As the hunter has a chance of hearing each time, a calculation will be done which will result in a percentage of the hunter hearing the player.</p>	<p>For hearing sound, I decided that I would need to create a probability of the hunter hearing the player based on realistic factors which will affect the probability such as the number of walls between the player and the hunter or the distance from the player to the hunter. This will all be used in a calculation in order to determine the chance of the hunter hearing the player.</p>
 <p>Sound needs to be simulated in a realistic way</p>	<p>For generating the sound, the main problem would be simulating it in a realistic way as I wanted the sound to be realistic in my program.</p>
 <p>Sound travels in all directions</p> <p>Sound needs to be simulated in a realistic way</p> <p>Loudness of sound changes depending on the number of objects inbetween the sound origin and the listener</p>	<p>For sound to be simulated realistically, I broke it down into 3 different problems, sound always originates from a point and travels in all directions so I needed to simulate that and in real life, the volume of the sound depends on factors such as the number of solid objects between the listener and the generator. The third problem will be explained further down</p>

<p>Sound travels in all directions</p> <p>Sound can be replicated using a circle</p>	<p>In order to simulate sound travelling in all directions, I decided that the sound can be replicated using a circle which originates from the player and a specific radius depending on the action the player is doing, for example, if the player is moving then there is a circle of a specific radius but if the player is not moving then there may not be any circle.</p>
<p>Loudness of sound changes depending on the number of objects inbetween the sound origin and the listener</p> <p>Need to check for the number of walls inbetween the hunter and the player</p>	<p>For the volume of the sound changing depending on the objects between the hunter and the player, I broke this down into another problem as I would need some way to detect the number of walls between the hunter and the player in order to determine the volume of the sound</p>

 <pre> graph TD A[Sound needs to be simulated in a realistic way] --> B[Sound changes depending on how fast you are moving] B --> C[Different sound coefficients depending on how fast the player is moving] </pre>	<p>This is the third problem from generating sound in a realistic way, in real life, when more work is being done to something, more sound is being produced, therefore, the sound should change depending on how fast the player is moving.</p>
 <pre> graph TD A[Player and hunter should not be able to walk through walls or other objects] --> B[Need to check for collisions] </pre>	<p>This is the third main problem for the physics aspect – I broke this problem down into checking for collisions, in order for anything to react to a collision, I need to know when a collision occurs</p>
 <pre> graph TD A[Need to check for collisions] --> B[Need to know the coordinates of both the objects] </pre>	<p>I broke this problem down into another problem because in order to know whether something is colliding, I need to know the positions of both of the objects so that I can figure out where the objects are relative to one another which would help determine the nature of the reaction when the objects collide.</p>



Round Ends:

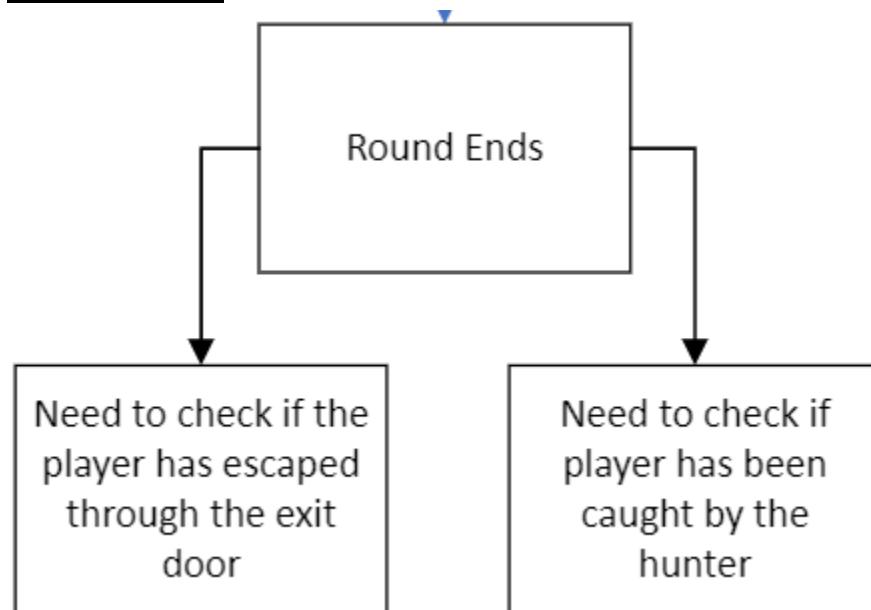
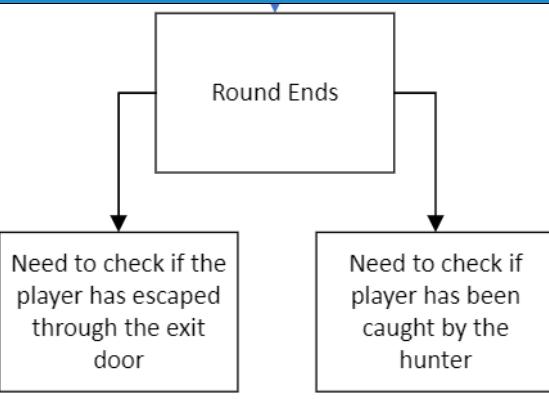


Diagram	Justification
 <pre>graph TD; A[Round Ends] --> B["Need to check if the player has escaped through the exit door"]; A --> C["Need to check if player has been caught by the hunter"]</pre>	<p>As there are only 2 situations which causes the round to end, I have split this problem down into 2 parts. I need to be able to check if the player has gone through the exit door as this would mean the player has won the game. If the player and the hunter have collided then the hunter has caught the player and the player loses the game.</p>

Describe the solution:

In this section I am going to be planning the classes for my program so that I can think ahead and decide how classes should interact, what classes there should be, what methods they need and attributes. I will also need to determine any extra variables or data structures which I will need for my program. All of the methods in each class also take self as a parameter although I haven't written that. Every class which inherits another class will also have all of the attributes and methods of the parent class, however, I am not going to rewrite the methods and attributes for the subclasses.

Extra variables:

- Lists for coordinates for each type of object in the map:
 - wood floor
 - concrete floor
 - carpet floor
 - lever wall
 - hiding space wall
 - empty wall
 - door
- Need variables for different colours which will be tuples containing 3 integers between 0 and 255 to make a specific colour, makes it easier to see what colours are being referenced in the program for different parts
- Need variables for each of the screen objects as they will be used in the main program, therefore, a variable is needed to carry out any actions on that object.

Classes

Class: Screen	Inherits: None
Parameters: colour – colour of screen (tuple) height – height of screen in pixels(int) width – width of screen in pixels(int) Attributes: colour – This is set as the parameter colour and is a tuple containing 3 elements which are integers from 0 – 255 controlling the colour of the screen (RGB)	Justification for class: This class is needed as I need to able to create multiple screens to be used in various situations since the user needs to be able to go to different screens and interact with them. Validation Discussion:

<p>texts – This is a 2D list which stores the text that needs to be displayed and the coordinates where they need to be displayed</p> <p>images - This is a 2D list which stores the images that need to be displayed and the coordinates where they need to be displayed</p> <p>buttons - This is a 2D list which stores the buttons that need to be displayed and the coordinates where they need to be displayed</p> <p>height – This is the height of the screen and is set as the parameter height which is an integer</p> <p>width – This is the width of the screen and is set as the parameter width which is an integer</p> <p>screen – This is the screen itself which uses a pygame method to create the screen, it is passed a tuple of the width and height of the screen</p>	I will need to validate the height, width and colour parameters passed into the class itself to ensure that they are in the correct format as this could cause an error in the program.
<p>Methods :</p> <p>displayImg() – This procedure displays an image on the screen</p> <p>renderMTexts(texts, sizes, colours, fonts, textsCoords) – This is a procedure which will render multiple lines of text</p> <p>renderText (textName, size, colour, font) – This is a function which renders a single line of text</p> <p>displayText () – This is a procedure that displays text</p> <p>createButton (id, image, x, y, scale) - This is a procedure which accesses the Button class to create a button and then attributes it to this specific class</p> <p>searchButton (id) – This is a function which searches and returns a button based on the ID of the button</p> <p>displayScreen () – This is a procedure which displays a new screen</p>	

addImages (images, coords, scale) – This is a procedure which appends all image based things into the image list

setColour (colour) – This is a procedure to set the colour of the screen

getHeight () – This is a function which returns the height of the screen

getWidth () – This is a function which returns the width of the screen

getScreen () – This is a function that returns the screen itself

Class: Button	Inherits: None
<p>Parameters:</p> <p>id – string to identify button (string)</p> <p>img – image used for the button (string)</p> <p>x – x coordinates of the button (int)</p> <p>y – y coordinates of the button (int)</p> <p>scale – the scale at which the image will be displayed (float)</p> <p>Attributes:</p> <p>id – This is the string passed to the class to help identify the button</p> <p>img – This is the image that will be displayed on the screen to represent the button which will be loaded using methods from the pygame library</p> <p>width – This will be the width of the button which will be calculated using a method from the pygame library</p> <p>height – This will be the height of the button which will be calculated using a method from the pygame library</p>	<p>Justification for class: This class is needed so that the user is able to navigate through the menu and be able to select different options depending on the button they press.</p> <p>Validation Discussion: For this class, not much validation is needed within methods themselves, however, for the parameters passed into the class for the first time, I need to make sure I validate the x and y coordinates and also the scale to check whether they are the right data type as these are the main values that can cause errors if they are not correct. Also the button cannot be clicked when it is not being displayed on the screen so I need to make sure that the user is not able to click the button</p>

transformedImg – This will be the image with the scale applied to it

rect – This will be the rectangle of the image of the button

clicked – This is whether or not the button has been clicked by the user.

Methods:

draw (screen) – This method will display the button on the screen using the parameter passed into the method

clickCheck (screen) - This method will check if the user has clicked on the button using pygame methods which will check if the users cursor has collided with the rect of the image and then checks if the user has clicked the left mouse button. Based on these conditions, a boolean value will be returned which determines whether the user has clicked the button or not.

Class: GameScreen

Parameters:

colour – This is the colour which the screen will be when displayed (tuple)

Attributes:

colour – This will be the colour which is passed in as a parameter

All other attributes are inherited from the screen

Methods:

displayGameScreen(map) – This will display the game screen using a pygame method and some other methods within the GameScreen class to display the map and other things. The map parameter will be in the form of a 2D list which will contain all of the objects in the map, this will be passed to another method within the class to display the map.

displayMap(map) - This will display the map on the screen and will take the map as the parameter which will be used in conjunction with other methods to check what objects should be displayed

Inherits: Screen

Justification for class:

This class is needed to display the screen for the game, a separate class is needed because the game screen will need some more unique methods

Validation Discussion:

For this class, the very first thing which needs to be validated is the colour passed into the class, as this can only be a tuple which contains 3 integer elements between 0 and 255. Some other things which can easily be validated and need to

<p>where. The map parameter will be a 2D list which will contain a list of all of the objects which are in the map and need to be displayed.</p> <p>displayObject(object) - This will take an object as a parameter (wall, door, floor) which will then be displayed on the screen using the coordinates in which it is meant to be. This will use another method to check what colour an object should be based on what object it is and what state it is in, for example, when the door is activated then it should look opened and when it is not activated then it should look closed.</p> <p>displayPlayer(player) - This will take the player object as a parameter and then display the player on the screen using a pygame method, it will also use a method from the player class to check if the player is hiding , because if the player is hiding then it should not be visible on the screen.</p> <p>displayHunter(hunter) - This will do a similar thing to the player and will display the hunter on the screen using a pygame method and take a hunter object as a parameter</p> <p>displayLeverCounter(player) - This will take a player object as a parameter and then use a method on the player object to get the number of levers which have been activated by the player and the maximum number of levers and then display a lever counter based on that.</p> <p>checkVisible(object) - This will be used in conjunction with the display object method and will take an object as the parameter and then use an object method to check if the object is visible or not, based on this, the colour of the object can be decided</p> <p>All other methods from screen superclass are also inherited</p>	<p>be are the player and hunter objects as within methods such as displayPlayer or displayHunter, I will need to make sure that the object passed into the method is of type player or hunter depending on the situation. Another place where validation is needed could be in display object as an if statement can be used with the or operator to check if the object passed into the method is one of those 3 objects which should be present in the map: wall, floor or door.</p>
---	--

Class: Object	Inherits: None
<p>Parameters: coords – This will be the coordinates of the object (tuple)</p> <p>Attributes: coords – This will be set as the variable coords that is passed as a tuple</p>	<p>Justification for class: This class is needed so that I can be more efficient and save time as these are the common attributes and methods which all of the objects in the map will need, therefore, instead of recoding the same thing again and again, I can use inheritance</p>

<p>rect – This will be the rectangle for the object which is a different rect type object created using the pygame library</p> <p>visible – This will be a boolean value and will determine whether an object is visible or not</p>	<p>and make the wall, floor, door, hidingSpace and lever class all inherit this class.</p> <p>Validation Discussion: In this class, the things which need to be validated are the coordinates which are passed to the class, as this could cause a problem if the coordinates are not the correct data type. Furthermore, I will also need to validate the value passed into the setVisible method as this must be a boolean value, otherwise it can cause problems when other methods need to check the visibility when it isn't a boolean value.</p>
<p>Methods:</p> <p>getCoords() - This will return the coordinates of the object relative to the map as a tuple containing 2 integers</p> <p>getRect() - This will return the rect attribute of the object</p> <p>getVisible() - This will return the visible attribute which will be a boolean value</p> <p>setVisible(visible) - This will set the visible attribute of the object and take the parameter visible – this will be a boolean value which will be set as the visible attribute.</p>	

Class: Wall	Inherits: Object
<p>Parameters:</p> <p>coords – This is the coords attribute which is inherited from the Object superclass (tuple)</p> <p>type – This will be used to determine what type of wall it should be and what item should be attached to it: hidingSpace, lever or nothing (string)</p> <p>Attributes:</p> <p>type – This is set as the parameter type which is passed into the class and will be used to determine what type of wall it is.</p> <p>item – This will use a method which will decide what object is attached to the wall, this will either be a hidingSpace, a lever or None</p> <p>Methods:</p>	<p>Justification for class: This class is needed as many different walls will need to be made and this will make it a lot easier to make multiple wall objects quickly.</p> <p>Validation Discussion: When the player is moving then</p>

Name: Muqtasid Zayyan Dar

Project title: Manhunt

<p>getItem() - This will return the item attribute</p> <p>setItem() - This will set the item attribute based on the value of type. This will be done by using decisions to check what the value of the attribute is and then depending on that, a specific type of object will be created</p>	
---	--

Class: Floor	Inherits: Object
<p>Parameters:</p> <p>coords – This is the coords attribute which is inherited from the Object superclass</p> <p>type – This will be used to determine what type of floor it should be which will determine its sound level (string)</p> <p>Attributes:</p> <p>type – This is set as the parameter type which is passed in as a string and will be used to determine what type of floor this will be set as</p> <p>sound – This will use a method to set the sound of the floor which will be a float value as the sound levels will be decimals less than 1</p> <p>Methods:</p> <p>setType() - This sets the type of floor depending on the string passed into the class</p> <p>setSound() - This will set the sound based on what string the type attribute is, similar to how the wall class works.</p> <p>getSound() - This will return the sound attribute of the floor which will be in the form of a float value</p>	<p>Justification for class:</p> <p>This class is needed as I will need multiple floor objects in my map, 1 for each coordinate which is covered by a floor, furthermore, there are multiple types of floor in my game so it would be a lot more efficient to have a class so I can create multiple floor objects quickly and easily</p> <p>Validation Discussion:</p> <p>For this class, I need to validate the string which is passed into the class in the setType() method as this could affect many other parts of my program and create errors if the wrong string is passed in.</p>

Class: HidingSpace	Inherits: Object
<p>Parameters:</p>	<p>Justification for class:</p>

Name: Muqtasid Zayyan Dar

Project title: Manhunt

<p>coords – This is the coords attribute which is inherited from the Object superclass</p> <p>Attributes:</p> <p>activationArea – This will be set as a rect which will be bigger than the hiding space but will not be displayed on the screen, this will be used to check whether the player is close enough to the hiding space so that the player can interact with it</p> <p>Methods:</p> <p>getActivationArea() - This will return the activationArea attribute in the form of a rect object.</p>	<p>This class is needed in order to have hiding space objects within my map and I need the activation area in order to be able to check if the player is close enough to the hiding space to be able to interact with it</p> <p>Validation Discussion:</p> <p>This class does not need any validation since there are no important data inputs which need to be checked</p>
--	--

Class: Lever	Inherits: Object
<p>Parameters:</p> <p>coords – This is the coords attribute which is inherited from the Object superclass</p> <p>Attributes:</p> <p>activated – This is a boolean value which is true or false and determines whether the lever has been activated or not</p> <p>activationArea – This is a rect object made using the pygame library which will not be displayed on the screen but will be used to determine if the player is close enough to the lever to interact with it</p> <p>Methods:</p> <p>getActivated() – This returns the activated attribute</p> <p>activate() – This sets the activated attribute as True</p> <p>deactivate() – This sets the activated attribute to False</p>	<p>Justification for class:</p> <p>This class is needed in order to have the lever objects within the map as there will be multiple. Furthermore, I will need to be able to check the state of each lever (activated or deactivated) so I created this class with those attributes</p> <p>Validation Discussion:</p> <p>In this class, I am going to use some form of indirect validation in order to be able to validate the data, this is done in the form of 2 different methods which are activate and deactivate, as the activated attribute can only be one of 2 values (since its boolean) I decided it would be easier to have 2 methods which do not take any parameters and based on the method, the value is changed to either true or false.</p>

Name: Muqtasid Zayyan Dar

Project title: Manhunt

getActivationArea() – This returns the activationArea attribute which is a rect type object	
--	--

Class: Door	Inherits: Object
<p>Parameters:</p> <p>coords – This is the coords attribute which is inherited from the Object superclass</p> <p>Attributes:</p> <p>activated – This is a boolean value which determines whether the door has been activated or not and will be set as False as the door starts as not being activated</p> <p>Methods:</p> <p>getActivated() – This returns the activated attribute</p> <p>activate() – This sets the activated attribute as True</p>	<p>Justification for class:</p> <p>This class is needed in order to have a door attribute in my map, furthermore, if I want to have multiple doors in my map then it will be easier to add in since I can easily create multiple door objects with this class. Also the doors need to have a state, as they can be activated after all of the levers have been activated.</p> <p>Validation Discussion:</p> <p>This class does not need any validation as there are set methods or data passed into the class which need to be validated.</p>

Class: Sprite	Inherits: None
<p>Parameters:</p> <p>x – This is the x coordinates on which the sprite will first spawn when it is displayed (int)</p> <p>y – This is the y coordinates on which the sprite will first spawn when it is first displayed (int)</p> <p>img – This is the image that will be used to display the sprite (string)</p> <p>speed – This will be the base speed at which the sprite will move across the screen (int)</p>	<p>Justification for class:</p> <p>This class is needed to be more efficient as I will have multiple classes with some of the same methods and attributes which will need to be recoded if I do not use this class whose attributes and methods will both be inherited by the subclasses. These classes are the player class and the hunter class as they are both a form of sprite. Furthermore, this means that if I need to make any changes for this class then it automatically is inherited in both other classes, however, if I did not use inheritance it would mean I would have to change the same thing in both classes which could</p>

<p>sprintMultiplier – This is the multiplier which will be used on the speed when the sprite needs to move faster (float)</p> <p>Attributes:</p> <p>x – This is set as the x parameter and will be used to control the location of the sprite on the screen</p> <p>y – This is set as the y parameter and will be used to control the location of the sprite on the screen</p> <p>img – This is set as the img parameter and will be used to display an image of the sprite on the screen</p> <p>hitbox – This is the hitbox of the sprite which will be a rect object the same size of the image</p> <p>speed – This is set as the speed parameter which will be used to control how many pixels the sprite moves across the screen</p> <p>sprintMultiplier – This is set as the sprint parameter and is the multiplier used to increase the speed by a certain factor in order to make the sprite move quicker across the screen</p> <p>forward – This will be a boolean value which will be set as false and will set as true while the sprite is moving forward</p> <p>backward – This will be a boolean value which will be set as false and will set as true while the sprite is moving backward</p> <p>left – This will be a boolean value which will be set as false and will be set as true while the sprite is moving to the left on the screen</p>	<p>cause logical errors since there are more points of failure to look out for</p> <p>Validation Discussion: For this class, the things which will need to be validated are the x and y parameters passed into the class as they must be within the map so that the sprite does not appear out of the map which could cause a logical error. I will also need to validate all of the attributes which can be accessed by set methods and make sure they are the correct data type as this could cause an error within the program. I also need to validate the sprite relative to other objects such as walls as the sprite needs to be able to collide with the walls since the sprites should not be able to walk through walls.</p>
--	---

right – This will be a boolean value which will be set as false and will be set as true while the sprite is moving to the right on the screen

sprint – This will be a boolean value which will be set as false and based on a condition which will be used in the subclasses, it can be set to true, in which case the sprintMultiplier will be used on the speed while moving the sprite

Methods:

displaySprite(screen) - This is to display the sprite on the screen and takes a screen object as a parameter, it will use the getCoords method to get the x and y attributes together as a coordinate and then use this along with the image for the sprite to display the sprite on the screen using pygame methods

moveForward() - This method will first set the forward attribute to true as whenever this method is invoked, it means the sprite is going to be moved forward. Then it will check if the sprint value is set to true or false and then based on that, if sprint is set to true then the y attribute will be decreased by the speed value with the sprint factor included. Otherwise, the y value will only be decreased by the value of speed. Finally, the setCoords method will be called to update the coordinates of the hitbox and the sprite

moveBackward() - This method will first set the backward attribute to true as whenever this method is invoked, it means the sprite is going to be moved backward. Then it will check if the sprint value is set to true or false and then based on that, if sprint is set to true then the y attribute will be increased by the speed value with the sprint factor included. Otherwise, the y value will only be increased by the value of speed. Finally, the setCoords method will be called to update the coordinates of the hitbox and the sprite

moveLeft() - This method will first set the left attribute to true as whenever this method is invoked, it means the sprite is going to be moved left. Then it will

check if the sprint value is set to true or false and then based on that, if sprint is set to true then the x attribute will be decreased by the speed value with the sprint factor included. Otherwise, the x value will only be decreased by the value of speed. Finally, the setCoords method will be called to update the coordinates of the hitbox and the sprite

moveRight() - This method will first set the right attribute to true as whenever this method is invoked, it means the sprite is going to be moved right. Then it will check if the sprint value is set to true or false and then based on that, if sprint is set to true then the x attribute will be increased by the speed value with the sprint factor included. Otherwise, the x value will only be increased by the value of speed. Finally, the setCoords method will be called to update the coordinates of the hitbox and the sprite

setSprint(value) - This method will set the sprint value of the sprite as true or false depending on what value is passed in, however, there needs to be an if statement which validates that the value passed in is a boolean value

getSprint() - This will return the sprint attribute

getHitbox() - This will return the hitbox attribute

getCoords() - This will return a tuple which contains the x attribute (coordinate) and then the y attribute (coordinate)

setCoords() - This will use the x and y attributes to then set the coordinates of the hitbox when it is called.

getForward() - This will return the value of the forward attribute

getBackward() - This will return the value of the backward attribute

<p>getLeft() - This will return the value of the left attribute</p> <p>getRight() - This will return the value of the right attribute</p> <p>checkCollision(map) - This method takes the map list as a parameter and then based on that, it will look through all of the objects within that list and check whether the sprite's hitbox has collided with any of those objects, if the object has collided with the sprite, then based on the direction in which the sprite was moving (this will be checked by seeing which of the movement attributes have been set to True) the sprite will be bounced back in the opposite direction by a certain number of pixels. This will be done by changing the relevant axial coordinate by a certain number of pixels.</p>	
---	--

Class: Player	Inherits: Sprite
<p>Parameters:</p> <p>x – Inherited and is the x coordinates on which the sprite will first spawn when it is displayed (int)</p> <p>y – Inherited and is the y coordinates on which the sprite will first spawn when it is first displayed (int)</p> <p>img – Inherited and is the image that will be used to display the sprite (string)</p> <p>speed – Inherited and is the base speed at which the sprite will move across the screen (int)</p> <p>sprintMultiplier – Inherited and is the multiplier which will be used on the speed when the sprite needs to move faster (float)</p> <p>Attributes:</p>	<p>Inherits: Sprite</p> <p>Justification for class: This class is needed as the user needs to be able to control their character on the screen and be able to interact with the game. Furthermore, the player needs to have specific behaviours and attribute, therefore, I need a class for the player to hold all of these attributes and behaviours.</p> <p>Validation Discussion: In this class there are 2 methods in which validation needs to occur, these methods are the set methods for the hiding and the sound attributes, I need to validate that the hiding attribute is a boolean value when it is set and also that the sound is within a specific range, this has not yet been decided but once it is, I will</p>

Name: Muqtasid Zayyan Dar

Project title: Manhunt

<p>hiding – This attribute will be a boolean value and will be used to determine whether the player is hiding or not</p> <p>sound – this will be a real or float value and will determine how much sound the player is making</p> <p>activatedLevers – This will be an integer and will be the number of levers which the player has activated</p> <p>sprintTimer – This will be an integer value which will determine the amount of time for which the player is able to sprint for</p>	need to add in a condition to check this. I will also need to validate other aspects of the player such as the sprinting time as the player is only allowed to sprint for a limited time before their sprint has to regenerate.
<p>Methods:</p> <p>displayPlayer() – This method will use the inherited method displaySprite to display the player, however, there will be another condition within this method so that the player is only displayed when they are not hiding</p> <p>checkForward() – This checks if the player is pressing the W key on the keyboard and if they are then the moveForward method is invoked</p> <p>checkBackward() – This checks if the player is pressing the S key on the keyboard and if they are then the move Backward method is invoked</p> <p>checkRight() - This checks if the player is pressing the D key on the keyboard and if they are then the moveRight method is invoked</p> <p>checkLeft() - This checks if the player is pressing the A key on the keyboard and if they are then the moveLeft method is invoked</p> <p>leverCheck() – This method will check through all of the lever objects and see if the player's hitbox is colliding with any of the lever objects, if it is then the player can interact with the lever and activate it, otherwise the player is not able to interact with the lever</p>	

hideCheck() – This method will check through all of the hidingSpace objects and see if the player's hitbox is colliding with any of the hidingSpace objects, if it is then the player can interact with the hidingSpace and the hiding attribute will be set to true which will change the behaviour of some of the methods. If the player is not colliding with any of the hidingSpace objects then they are not able to interact with them.

interact() – This method will check if the player is pressing the E button on the keyboard and if the player is then it will invoke both the leverCheck and hideCheck methods to then check if the player is trying to interact with anything.

sprintCheck() – This method checks if the player is pressing the shift button and if the player is pressing the shift button then the sprint attribute is set to true, if they are not then the sprint attribute is set to false

setSound(value) – This method takes in a value as a parameter and then sets this value as the player's sound attribute

getSound() – This method will return the sound attribute for the player

activateLever() – This method will increment the activatedLevers attribute by 1

getActivatedLevers() – This will return the activatedLevers attribute

getHiding() – This will return the hiding attribute

setHiding(value) – This set the hiding attribute as the value parameter passed in which needs to be a boolean value

checkWin – This method will check if the number of levers the player has activated is equal to the number of levers in the level and then based on this, the player will be allowed to

<p>walk through the doors, this method will also check when the player walks through the doors, if the player walk through then it will end the game</p> <p>fov – This method will use the sightProjectiles and then launch them in all directions and then after all of the sightProjectiles have collided with walls then this method will check what objects each of the projectiles have collided with and then make these objects visible to the player.</p>	
--	--

Class: Hunter	Inherits: Sprite
<p>Parameters:</p> <p>x – Inherited and is the x coordinates on which the sprite will first spawn when it is displayed (int)</p> <p>y – Inherited and is the y coordinates on which the sprite will first spawn when it is first displayed (int)</p> <p>img – Inherited and is the image that will be used to display the sprite (string)</p> <p>speed – Inherited and is the base speed at which the sprite will move across the screen (int)</p> <p>sprintMultiplier – Inherited and is the multiplier which will be used on the speed when the sprite needs to move faster (float)</p> <p>Attributes:</p> <p>chase – This will be a boolean value and is a state for if the hunter is chasing the player or not, if the hunter is chasing the player then this is set to True – based on this, the sprint multiplier will be applied to the normal speed</p>	<p>Justification for class:</p> <p>This class is needed so that I can have a hunter in my game which needs to be able to move around and chase the player. It also needs to be able to path find around the map so that it can do things like chase the player or also create random paths in order to follow. Also the hunter needs to be able to hear the player so the hunter class needs to carry all of this out.</p> <p>Validation Discussion:</p> <p>In this class, there are many things which need to be validated. Firstly, the data passed into all of the set methods need to be validated, for all of the cost attributes, the values cannot be negative or greater than the diagonal length of the map. For the chase and heard attributes, the values need to be checked to make sure they are boolean values passed into the set methods. Another thing which needs to be validated are the paths generated</p>

hCost – This will be the heuristic cost for the hunter when using the A* algorithm which is an approximation of how far the hunter is from reaching the goal coordinates

gCost – This will be the g cost when using the A* algorithm which will be a measurement of how far the hunter is from the start coordinates

fCost – This will be the value of the gCost plus the hCost and is used as an estimate to the overall length of the path

heard – This attribute will be a boolean value and will be used to check if the hunter has heard a sound

Methods:

aStar – This will be the A* algorithm which will be used to calculate a path which is good enough for the hunter to follow in the map.

calcHValue – This method will use the coordinates of the goal and the starting coordinates (which will be the hunters coordinates) in order to calculate the heuristic value for the distance from the hunter to the goal

calcGValue() - This method will calculate the g cost of the hunter and then set the result as the gCost attribute

calcFCost() - This method will add the hCost attribute and the gCost attribute after updating them and then set this value as the fCost attribute

setHCost – This method will take a value and set it as the hCost attribute

setGCost – This method will take a value and set it as the gCost attribute

setFCost – This method will take a value and set it as the fCost attribute

with the A* algorithm, this is because the hunter should not be able to travel where there is a wall so that should not count as a visitable node in the A* algorithm. For the sound method, I need to validate the walls since I only need to count the wall objects when the sound projectile collides with a wall

Name: Muqtasid Zayyan Dar

Project title: Manhunt

getFCost – This method returns the fCost attribute

getHCost – This method returns the hCost attribute

getGCost – This method returns the gCost attribute

fov – This method will use the sightProjectile objects in order to create a fov in whatever direction the hunter is moving, it will only be in the direction in front of the hunter

sound – This method will determine whether the hunter is able to hear the player or not and if they are able to hear the player then if this happens then it will return coordinates for the hunter to follow from where the sound came from and heard will be set to True

checkWalls – This method will check the number of walls between the player and the hunter using a sound projectile object and then return the number of walls

randomPath – This method will generate random valid paths which the hunter can follow in the map for when the hunter is not actively chasing the player

goalPath – This method will generate a path to a specific goal, whether that is to the origin of a sound at some coordinates or the coordinates for the player (when the hunter has seen the player)

Class: Projectile	Inherits: Physics
<u>Parameters:</u>	<u>Justification for class:</u>
xSpeed – this will be used to determine the speed of the projectile in the x direction(int)	This class is needed as I will need different types of projectiles in my game so I need to create a general projectile class which both of these other classes can inherit and use. This makes it more
ySpeed – This will be used to determine the speed of the projectile in the y direction (int)	

<p>length – This will be used to determine the size of the projectile (int)</p> <p>coords – This will be the coordinates which the projectile will originate from</p> <p>Attributes:</p> <p>xSpeed – This will be set as the parameter xSpeed which is passed into the class and will be the number of pixels the projectile will be moved in the x direction</p> <p>ySpeed – This will be set as the parameter ySpeed which is passed into the class and will be the number of pixels the projectile will be moved in the y direction</p> <p>length – This will be the length parameter passed into the class</p> <p>collided – This will be set as a boolean value and will be used with a specific condition in each sub class to determine whether the projectile has collided with a specific object yet</p> <p>projectile – This will be a rect object which will be created</p> <p>coords – This will be the coordinates where the projectile will appear from and these will be changed using the x speed or y speed of the projectile</p>	<p>efficient for me as I will not need to recode the same things for multiple projectiles classes.</p> <p>Validation Discussion:</p> <p>For this class, validation is needed for a few things such as the set methods for the speed and collided attributes, this is because the collided attribute can only be a boolean value and if it is not, it can cause an error in my program. For the speed attributes, they can only be values from negative half the length of each object to positive half the length of each object. This is because if the speed is below or above each of those values, then each time the placement of the projectile updates, it may skip some objects which can cause errors with detecting collisions between the projectile and the objects. Another thing which needs to be validated is the collisions as I need to know when a collision occurs between a specific object and the projectile, therefore, this validation will be done in the sub classes as it is specific to each one.</p>
<p>Methods:</p> <p>moveProjectile() – This method will update both of the x and y values of the projectiles using the speed attributes for the projectiles</p> <p>setCollided(value) – This will set the collided attribute as the value parameter passed into the method</p> <p>getCollided() – This will return the collided attribute</p> <p>setXSpeed(value) – This will set the xSpeed attribute as the value parameter passed into the method</p>	

<p>setYSpeed(value) – This will set the ySpeed attribute as the value of the parameter passed into the method</p> <p>collideCheck(object) – This will use a pygame method to check if the object passed into the method has collided with the rect of the projectile</p>	
--	--

Class: SightProjectile	Inherits: Projectile
<p>Parameters:</p> <p>xSpeed – This is the x speed of the projectile in pixels (int)</p> <p>ySpeed – This is the y speed of the projectile in pixels (int)</p> <p>length – This is the length for the width and height of the projectile (int)</p> <p>coords – These are the coordinates which the projectile will originate from (tuple)</p> <p>Attributes:</p> <p>collided – This will be set as an empty list and will hold all of the objects which the projectile has collided with</p>	<p>Justification for class:</p> <p>This class is needed for both the hunter and the player because both of these sprites need to have an FOV so that the player is able to see specific parts of the map on the screen and the hunter is able to look around for the player and detect them.</p> <p>Validation Discussion:</p> <p>For this class, there are a few things which need to be validated, I need to validate the type of objects which the projectiles collide with, this is because a different action needs to be carried out if the projectile has collided with the floor compared to if the projectile has collided with the player, if I do not validate this then I will not be able to know what objects are the floor and which objects is the player.</p>
<p>Methods:</p> <p>wallCollideCheck() - This method will check if the projectile has collided with a wall and then when the projectile has collided with a wall then it will set the collided attribute as True</p> <p>playerCollideCheck() - This method will check if the projectile has collided with the player and when this happens, it will set the collided attribute as True</p>	

Name: Muqtasid Zayyan Dar

Project title: Manhunt

collisionCheck() - This method will check through all of the objects in the map and check if the object has collided with the projectile, if it has then the visibility of that object is set to True, this method will run as long as collided attribute for the projectile is false

Class: SoundProjectile	Inherits: Projectile
<p>Parameters:</p> <p>xSpeed – This will be used to control the x speed of the projectile (int)</p> <p>ySpeed – This will be used to control the y speed of the projectile (int)</p> <p>length – This will be used for the height and width of the projectiles (int)</p> <p>coords – This will be the origin coordinates for the projectile (tuple)</p> <p>Attributes:</p> <p>wallNum – This attribute will be used to check the number of walls which the sound projectile has collided with</p>	<p>Justification for class:</p> <p>This class is needed in order for the hunter to be able check the number of walls between it and the player, therefore, some different methods are also needed as this class is different to the previous projectile class.</p> <p>Validation Discussion:</p> <p>In this class, I will mainly need to validate the objects which the projectile is colliding with since the projectile needs to know whether the object is a player type object, a wall type object or a different object in order to carry out the relevant task</p>
<p>Methods:</p> <p>wallCollideCheck – This method will check if the projectile has collided with a wall and if it has then it will increment wallNum attribute by 1 using incrementWalls method</p> <p>incrementWalls – This will increment the wallNum attribute by 1</p> <p>playerCollisionCheck – This will check if the projectile has collided with the player and if it has then it will set the collided attribute for the projectile as True</p>	

getWallNum – This method will return the wallNum attribute	
--	--

Class: Map	Inherits: Projectile
<p>Parameters:</p> <p>height – This will be the height of the map (int)</p> <p>width – This will be used to determine the width of the map (int)</p> <p>floorCoords – This will be the coordinates of all of the floors in the game and will be a dictionary holding lists for each type of floor (dict)</p> <p>wallCoords – This will be the coordinates for all of the different walls in the game and will be a dictionary holding lists for each type of wall (dict)</p> <p>doorCoords – This will be the coordinates for the door (tuple)</p> <p>Attributes:</p> <p>height – This will be set as the height parameter</p> <p>width – This will be set as the width parameter</p> <p>floorCoords – This will be set as the floorCoords parameter</p> <p>wallCoords – This will be set as the wallCoords parameter</p> <p>doorCoords – This will be set as the doorCoords parameter</p> <p>map – This will be a 2D array which will hold all of the objects which will be present in the map in the correct positions, the number of arrays in the array will be</p>	<p>Justification for class:</p> <p>This class is needed so that I can create a grid system for the objects which will be in the game, it will also help me keep track of the hunter and player's location relative to other things when needed. Furthermore, this class is also needed so that I can use the A* algorithm for the hunter because without a proper grid system I cannot use the A* algorithm</p> <p>Validation Discussion:</p> <p>In this class, I will need to validate a few things. Firstly, when the map is created, each coordinate for the map needs to be validated as coordinates cannot exist outside of the map. When the getObject method is used then the coordinates passed in need to be validated as it must be within the map. The getWalls and getFloor methods also need to be validated as it can only return the floor and wall types which actually exist.</p>

decided by the height attribute and the number of objects in each array will be decided by the width attribute

Methods:

`createMap()` - This method will use the height and width attributes along with the different coordinate attributes to create the map and then set the result as the map attribute.

`getObject(coords)` - This method will use the coordinates passed into the method and the map attribute in order to return the object located in that position on the map.

`getWalls(type)` - This method will use the parameter type in order to return a list of a certain type of wall, if type is not passed into the method then it will just return all of the wall objects.

`getFloors(type)` - This method will use the parameter type in order to return a list of a certain type of floor, if type is not passed into the method then it will just return all of the floor objects

`getDoor()` - This method will return the door object using the `doorCoords` and `getObject` method

`getAllObjects()` - This method will take the map and append every object to a 1D array and then return that array

`getMap()` - This method will return the map attribute

`addFloors ()` - This method will create all of the floor objects and add them to the corresponding positions on the map

Name: Muqtasid Zayyan Dar

Project title: Manhunt

addWalls () - This method will create all of the wall objects and add them to the corresponding positions on the map

addDoor () - This method will create the door object and add them to the corresponding positions on the map

Pseudocode:**Screen class:****CLASS Screen ()**

private colour

private height

private width

private images

private buttons

private texts

public PROCEDURE constructor (screenColour, screenHeight, screenWidth)

colour = screenColour

height = screenHeight

width = screenWidth

buttons = []

images = []

texts = []

ENDPROCEDURE**private PROCEDURE renderMTexts (allTexts, sizes, colours, fonts, textCoords)**

textList = []

```
counter = 0

WHILE counter != len (texts)

    render = renderText (texts[counter], sizes[counter], colours[counter], fonts[counter])

    textList.append (render)

    counter = counter + 1

ENDWHILE

texts.append (textList)

texts.append (textCoords)

ENDPROCEDURE

private FUNCTION renderText (textName, size, colour, font)

textFont = pygame.font (font, size)

text = textFont.render (textName, colour)

return text

ENDFUNCTION

private PROCEDURE displayText ()

textIndex = 0

WHILE textIndex != len (texts[0])

    displayText (texts[0][textIndex], texts[1][textIndex])

ENDWHILE
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

ENDPROCEDURE

```
private PROCEDURE displayImg ()  
    imageIndex = 0  
    WHILE imageIndex != len(images[0])  
        image = pygame.loadImage (images[0][imageIndex])  
        width = image.getWidth ()  
        height = image.getHeight ()  
        pygame.displayImage (image, images[1][imageIndex])  
        imageIndex = imageIndex + 1  
    ENDWHILE
```

ENDPROCEDURE

```
public PROCEDURE createButton (buttonID, buttonImage, x, y, buttonScale)  
    button = Button (buttonID, buttonImage, x, y, buttonScale)  
    buttons.append (button)
```

ENDPROCEDURE

```
public FUNCTION searchButton (buttonID)  
    FOR button IN buttons:  
        IF button.getID = buttonID THEN
```

```
    RETURN button  
ENDIF  
ENDFOR  
ENDFUNCTION
```

```
public PROCEDURE displayScreen ()  
    pygame.displayScreen ()  
    displayImg ()  
    displayText  
ENDPROCEDURE
```

```
public PROCEDURE addImages (images, imageCoords, imageScales)  
    images.append (images)  
    images.append (imageCoords)  
    images.append (imageScales)  
ENDPROCEDURE
```

```
public PROCEDURE setColour (screenColour)  
    colour = screenColour  
ENDPROCEDURE
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

public FUNCTION getHeight ()

 RETURN height

ENDFUNCTION

public FUNCTION getWidth ()

 RETURN width

ENDFUNCTION

public FUNCTION getScreen ()

 RETURN screen

ENDFUNCTION

ENDCLASS

Button class:

CLASS Button ()

 private id

 private image

 private width

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
private height  
private rect  
private coords  
private clicked  
private transformedImage
```

```
public PROCEDURE constructor (buttonID, image, x, y, scale)  
    id = buttonID  
    image = pygame.loadImage (image)  
    width = image.getWidth ()  
    height = image.getHeight ()  
    transformedImage = pygame.transform(image, width * scale, height * scale)  
    rect = transformedImage.getRect ()  
    coords = (x, y)  
    clicked = False  
ENDPROCEDURE
```

```
private PROCEDURE draw (screen)  
    pygame.displayImage (transformedImage, coords)  
ENDPROCEDURE
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
private FUNCTION clickCheck (screen)
    draw (screen)
        hasClicked = False
        mousePosition = pygame.getMousePos ()
        IF rect collide with mousePosition THEN
            IF left mouse button pressed THEN
                clicked = True
                hasClicked = True
            ENDIF
        ENDIF
        IF left mouse button NOT pressed THEN
            clicked = False
        ENDIF
    RETURN hasClicked
ENDFUNCTION
```

ENDCLASS

GameScreen class:

```
CLASS GameScreen (INHERITS: Screen)
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

//This class does not need another constructor as it does not have any extra attributes which need declaring since all attributes used are inherited from screen

```
public PROCEDURE displayGameScreen (map, player, hunter)
    displayScreen ()
    displayMap (map)
    displayPlayer (player)
    displayHunter (hunter)
    displayLeverCounter (player)
ENDPROCEDURE
```

```
private PROCEDURE displayMap (map)
    FOR row IN map:
        FOR object IN row
            IF checkVisible (object) = True THEN
                displayObject (object)
            ENDIF
        ENDFOR
    ENDFOR
ENDPROCEDURE
```

```
private PROCEDURE displayObject (object)
```

```
    pygame.displayRect (object)
```

```
ENDPROCEDURE
```

```
private PROCEDURE displayPlayer (player)
```

```
    player.displayPlayer ()
```

```
ENDPROCEDURE
```

```
private PROCEDURE displayHunter (hunter)
```

```
    hunter.displayHunter ()
```

```
ENDPROCEDURE
```

```
private PROCEDURE displayLeverCounter (player)
```

```
    activated = player.getActivatedLevers ()
```

```
    text = activated + "/ 7"
```

```
    pygame.displayText (text)
```

```
ENDPROCEDURE
```

```
private FUNCTION checkVisible (object)
```

```
    RETURN object.getVisible ()
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

ENDFUNCTION

ENDCLASS

Object class:

CLASS Object ()

 private coords

 private rect

 private visible

public PROCEDURE constructor (objectCoords, height, width)

 coords = objectCoords

 rect = pygame.rect(coords, height, width)

 visible = False

ENDPROCEDURE

public FUNCTION getCoords ()

 RETURN coords

ENDFUNCTION

public FUNCTION getRect ()

Name: Muqtasid Zayyan Dar

Project title: Manhunt

RETURN rect

ENDFUNCTION

public FUNCTION getVisible ()

RETURN visible

ENDFUNCTION

public PROCEDURE setVisible (value)

visible = value

ENDPROCEDURE

ENDCLASS

Wall class:

CLASS Wall (INHERITS: Object)

private item

private type

public PROCEDURE constructor (wallType)

type = wallType

item = setItem (wallType)

Name: Muqtasid Zayyan Dar

Project title: Manhunt

ENDPROCEDURE

public FUNCTION getItem ()

 RETURN item

ENDFUNCTION

private ENDFUNCTION setItem ()

 IF type = 0 THEN

 RETURN none

 ELIF type = 1 THEN

 RETURN Lever (coords[0], coords[1], height, width)

 ELIF type = 2 THEN

 RETURN HidingSpace (coords[0], coords[1], height, width)

 ELSE:

 OUTPUT "NOT VALID NUM FOR WALL ITEM"

ENDIF

ENDFUNCTION

ENDCLASS

Floor class:

Name: Muqtasid Zayyan Dar

Project title: Manhunt

CLASS Floor (INHERITS: Object)

private type

private sound

public PROCEDURE constructor (floorType)

 type = floorType

 sound = setSound (floorType)

ENDPROCEDURE

private FUNCTION setSound ()

 IF type = 0 THEN

 RETURN 0.3

 ELIF type = 1 THEN

 RETURN 0.6

 ELIF type = 2 THEN

 RETURN 0.9

 ELSE:

 OUTPUT "NOT VALID NUM FOR FLOOR SOUND"

 ENDIF

ENDFUNCTION

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
public PROCEDURE getSound ()  
    RETURN sound  
  
ENDPROCEDURE  
  
ENDCLASS
```

Lever class:

CLASS Lever (INHERITS: Object)

```
private activationArea
```

```
private activated
```

```
public PROCEDURE constructor (x, y, height, width)  
    activationArea = pygame.rect (x, y, height * 3, width * 3)  
    activated = False  
  
ENDPROCEDURE
```

```
public PROCEDURE activate ()  
    activated = True  
  
ENDPROCEDURE
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
public PROCEDURE deactivate ()
```

```
    activated = False
```

```
ENDPROCEDURE
```

```
public FUNCTION getActivated ()
```

```
    RETURN activated
```

```
ENDFUNCTION
```

```
ENDCLASS
```

HidingSpace class:

```
CLASS HidingSpace (INHERITS: Object)
```

```
    private activationArea
```

```
    public PROCEDURE constructor (x, y, height, width)
```

```
        activationArea = pygame.rect (x, y, height * 3, width * 3)
```

```
    ENDPROCEDURE
```

```
ENDCLASS
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Door class:

CLASS Door (INHERITS: Object)

 private activated

public PROCEDURE constructor ()

 activated = False

ENDPROCEDURE

public FUNCTION getActivated ()

 RETURN activated

ENDFUNCTION

public PROCEDURE activated ()

 activated = True

ENDPROCEDURE

ENDCLASS

Sprite class:

Name: Muqtasid Zayyan Dar

Project title: Manhunt

CLASS Sprite ()

```
private x  
private y  
private image  
private hitbox  
private speed  
private sprintMultiplier  
private forward  
private backward  
private left  
private right  
private sprint
```

public PROCEDURE constructor (xCoord, yCoord, spriteImage, spriteSpeed, spriteSprintMultiplier)

```
x = xCoord  
y = yCoord  
image = spriteImage  
hitbox = image.getRect ()  
speed = spriteSpeed  
sprintMultiplier = spriteSprintMultiplier
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

forward = False

backward = False

left = False

right = False

sprint = False

ENDPROCEDURE

public PROCEDURE displaySprite ()

 pygame.displayImage (image, (x, y))

ENDPROCEDURE

private PROCEDURE moveForward ()

 forward = True

 IF sprint = True THEN

 y = y - (speed * sprintMultiplier)

 ELSE

 y = y - speed

 ENDIF

 setCoords ()

ENDPROCEDURE

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
private PROCEDURE moveBackward ()
    backward= True
    IF sprint = True THEN
        y = y + (speed * sprintMultiplier)
    ELSE
        y = y + speed
    ENDIF
    setCoords ()
ENDPROCEDURE
```

```
private PROCEDURE moveLeft ()
    left = True
    IF sprint = True THEN
        x = x - (speed * sprintMultiplier)
    ELSE
        x = x - speed
    ENDIF
    setCoords ()
ENDPROCEDURE
```

```
private PROCEDURE moveRight ()
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
right = True  
IF sprint = True THEN  
    x = x + (speed * sprintMultiplier)  
ELSE  
    x = x + speed  
ENDIF  
setCoords ()  
ENDPROCEDURE  
  
private PROCEDURE setSprint (value)  
    sprint = value  
ENDPROCEDURE  
  
private FUNCTION getHitbox ()  
    RETURN hitbox  
ENDFUNCTION  
  
private FUNCTION getCoords ()  
    RETURN (x, y)  
ENDFUNCTION
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

private PROCEDURE setCoords ()

 hitbox.center = (x, y)

ENDPROCEDURE

private FUNCTION getForward ()

 RETURN forward

ENDFUNCTION

private FUNCTION getBackward()

 RETURN backward

ENDFUNCTION

private FUNCTION getLeft ()

 RETURN left

ENDFUNCTION

private FUNCTION getRight ()

 RETURN right

ENDFUNCTION

private FUNCTION checkCollision (map)

Name: Muqtasid Zayyan Dar

Project title: Manhunt

FOR row IN map

FOR object IN row

IF hitbox.collideCheck (object) THEN

IF getForward () = True THEN

moveBackward ()

ENDIF

IF getBackward () = True THEN

moveForward ()

ENDIF

IF getLeft () = True THEN

moveRight ()

ENDIF

IF getRight () = True THEN

moveLeft ()

ENDIF

ENDIF

ENDFOR

ENDFOR

ENDFUNCTION

Name: Muqtasid Zayyan Dar

Project title: Manhunt

ENDCLASS

Player class:

CLASS Player (INHERITS: Sprite)

```
private hiding  
private sound  
private activatedLevers  
private sprintTimer
```

public PROCEDURE constructor (playerSprintTime)

```
    hiding = False  
    sound = 0  
    activatedLever = 0  
    sprintTimer = playerSprintTime
```

ENDPROCEDURE

public PROCEDURE displayPlayer ()

```
    IF hiding = False THEN  
        displaySprite ()  
    ENDIF
```

ENDPROCEDURE**private PROCEDURE checkForward (keyPressed)**

IF keyPressed = W THEN

moveForward ()

ELSE

forward = False

ENDIF

ENDPROCEDURE**private PROCEDURE checkBackward (keyPressed)**

IF keyPressed = S THEN

moveBackward ()

ELSE

backward = False

ENDIF

ENDPROCEDURE**private PROCEDURE checkLeft (keyPressed)**

IF keyPressed = A THEN

moveLeft ()

Name: Muqtasid Zayyan Dar

Project title: Manhunt

ELSE

left = False

ENDIF

ENDPROCEDURE

private PROCEDURE checkRight (keyPressed)

IF keyPressed = D THEN

moveRight()

ELSE

right = False

ENDIF

ENDPROCEDURE

private PROCEDURE leverCheck (item)

IF item = Lever THEN

IF item.getActivated () = False THEN

activateLever ()

item.activate ()

ENDIF

ENDIF

ENDPROCEDURE

```
private PROCEDURE hideCheck (item)
```

```
    IF item = HidingSpace THEN
```

```
        setHiding (True)
```

```
ENDPROCEDURE
```

```
private PROCEDURE interact (keyPress, map)
```

```
    IF keyPress = E THEN
```

```
        walls = map.getItemWalls
```

```
        FOR wall IN itemWalls
```

```
            item = wall.getItem ()
```

```
            IF hitbox.collideCheck (item.getActivationArea ()) THEN
```

```
                hideCheck (item)
```

```
                leverCheck (item)
```

```
            ENDIF
```

```
        ENDFOR
```

```
    ENDIF
```

```
ENDPROCEDURE
```

```
private PROCEDURE sprintCheck (keyPress)
```

```
    IF keyPress = SHIFT THEN
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
sprint = True  
ELSE  
    sprint = False  
ENDIF  
ENDPROCEDURE  
  
private PROCEDURE setSound (map)  
    object = map.getObject ((x, y))  
    IF object = Floor THEN  
        sound = object.getSound ()  
    ENDIF  
ENDPROCEDURE  
  
public FUNCTION getSound ()  
    RETURN sound  
ENDFUNCTION  
  
private PROCEDURE activateLever ()  
    activatedLevers = activatedLevers + 1  
ENDPROCEDURE
```

```
private FUNCTION getActivatedLevers ()
```

```
    RETURN activatedLevers
```

```
ENDFUNCTION
```

```
private FUNCTION getHiding ()
```

```
    RETURN hiding
```

```
ENDFUNCTION
```

```
private PROCEDURE setHiding (value)
```

```
    hiding = value
```

```
ENDPROCEDURE
```

```
private FUNCTION checkWin (map)
```

```
    IF activatedLevers = map.getMaxLevers () THEN
```

```
        RETURN True
```

```
    ELSE
```

```
        RETURN False
```

```
    ENDIF
```

```
ENDFUNCTION
```

```
private PROCEDURE fov (map)
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
allObjects = map.getAllObjects ()  
FOR object IN allObjects  
    object.setVisible (False)  
ENDFOR  
  
IF projectiles have not been created yet THEN  
    create 8 SightProjectiles  
    set speed of each SightProjectile in 8 different directions  
ENDIF  
  
FOR each SightProjectile  
    launch SightProjectile from player coordinates  
ENDFOR  
  
FOR each SightProjectile  
    collided = get list of objects which projectile has collided with  
    FOR object IN collided  
        object.setVisible (True)  
    ENDFOR  
ENDFOR  
  
ENDPROCEDURE  
  
ENDCLASS
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Hunter class:

CLASS Hunter (INHERITS: Sprite)

 private chase

 private hCost

 private gCost

 private fCost

 private heard

public PROCEDURE constructor ()

 chase = False

 hCost = 0

 gCost = 0

 fCost = 0

 heard = False

ENDPROCEDURE

private FUNCTION aStar (map, endCoords)

 open = []

 closed []

 endNode = map.getObject (endCoords)

 startNode = map.getObject (getCoords)

```
open.append (startNode)

WHILE found = False

    current = node in open with lowest f_cost

    remove current from open

    add current to closed

    IF current = endNode THEN

        found = True

    ENDIF

    FOR each neighbour of current node

        IF neighbour is not traversable or neighbour is in closed THEN

            skip to next neighbour

        ENDIF

        IF new path to neighbour is shorter OR neighbour is NOT in open THEN

            calcFCost ()

            set parent of neighbour to current

            IF neighbour is NOT in open THEN

                open.append (neighbour)

            ENDIF

        ENDIF

    ENDFOR

ENDWHILE
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

RETURN current

ENDFUNCTION

private PROCEDURE calcHValue (endCoords)

startCoords = getCoords ()

xDist = startCoords[0] - endCoords[0]

yDist = startCoords[0] - endCoords[0]

xSqr = square xDist

ySqr = square yDist

hValue = square root (xSqr + ySqr)

hCost = hValue

ENDPROCEDURE

private PROCEDURE calcGValue (current)

gCost = gCost + 1

ENDPROCEDURE

private PROCEDURE calcFCost ()

fCost = hCost + gCost

ENDPROCEDURE

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
private PROCEDURE setHCost (value)
```

```
    hCost = value
```

```
ENDPROCEDURE
```

```
private PROCEDURE setGCost (value)
```

```
    gCost = value
```

```
ENDPROCEDURE
```

```
private PROCEDURE setFCost (value)
```

```
    fCost = value
```

```
ENDPROCEDURE
```

```
private FUNCTION getHCost ()
```

```
    RETURN hCost
```

```
ENDFUNCTION
```

```
private FUNCTION getFCost ()
```

```
    RETURN fCost
```

```
ENDFUNCTION
```

```
private FUNCTION getGCost ()
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

RETURN gCost

ENDFUNCTION

private PROCEDURE fov (map, player)

allWalls = map.getWalls ()

projectiles = []

IF projectiles have not yet been created THEN

 create 3 SightProjectiles and add to projectiles list

ENDIF

get direction hunter is moving in

change xSpeed and ySpeed of projectile based on the direction

FOR projectile in projectiles

 IF projectile has collided or has just been created THEN

 setCollided (False)

 colided = launch the projectile and get whether or not the projectile has collided with the player

 ENDIF

ENDFOR

IF collided = True THEN

 coords = player.getCoords ()

 aStar (coords)

ENDIF

ENDPROCEDURE

```
private PROCEDURE sound (player, map)
    sound = 100
    soundMult = player.getSound ()
    IF player is moving THEN
        sound = soundMult * sound
        wallNum = checkWalls (player, map)
        sound = sound - wallNum
        IF sound < 50 THEN
            heard = False
        ELIF sound <= 75 THEN
            randNum = generate random number between 50, 75
            IF randNum = sound THEN
                heard = True
            ELSE
                heard = False
            ENDIF
        ELSE
            heard = True
        ENDIF
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

ENDIF

ENDPROCEDURE

private PROCEDURE checkWalls (player, map)

soundProjectile = create SoundProjectile and set speed to 0

soundProjectile.setCollided (False)

pCoords = player.getCoords ()

hCoords = hunter.getCoords ()

coordDistance = use the pCoords and hCoords to calculate the distance of the player relative to the hunter

angle = use trigonometry and the coordDistance to calculate the angle at which the projectile needs to be launched

speedComp = use trigonometry on the angle and the speed in order to calculate the x and y components of the speed

soundProjectile.setXSpeed (speedComp[0])

soundProjectile.setYSpeed (speedComp[1])

wallNum = soundProjectile.getWallNum ()

RETURN wallNum

ENDPROCEDURE

private PROCEDURE randomPath ()

randCoords = generate random coordinates within map boundaries

aStar (randCoords)

ENDPROCEDURE

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
private PROCEDURE goalPath (goalCoords)
    aStar (randCoords)
ENDPROCEDURE

ENDCLASS
```

Projectile class:

```
CLASS Projectile ()
    private xSpeed
    private ySpeed
    private length
    private collided
    private projectile
    private coords
```

```
public PROCEDURE constructor (projXSpeed, projYSpeed, projLen, projCoords)
    xSpeed = projXSpeed
    ySpeed = projYSpeed
    length = projLen
    collided = False
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
coords = projCoords
projectile = pygame.rect (projCoords, length, length) //length is written twice as the projectile will have the same height and width which I just decided to call length to make it easier
```

ENDPROCEDURE

private PROCEDURE moveProjectile ()

```
rect.x = rect.x + xSpeed
rect.y = rect.y +ySpeed
```

ENDPROCEDURE

private PROCEDURE setCollided (value)

```
collided = value
```

ENDPROCEDURE

private FUNCTION getCollided ()

```
RETURN collided
```

ENDFUNCTION

private PROCEDURE setXSpeed (value)

```
xSpeed = value
```

ENDPROCEDURE

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
private PROCEDURE setYSpeed (value)
```

```
    ySpeed = value
```

```
ENDPROCEDURE
```

```
private FUNCTION checkCollision(object)
```

```
    rect.collideCheck (object)
```

```
ENDFUNCTION
```

```
ENDCLASS
```

SightProjectile class:

```
CLASS SightProjectile (INHERITS: Projectile)
```

```
    private collided
```

```
    public PROCEDURE constructor ()
```

```
        collided = False
```

```
    ENDPROCEDURE
```

```
    private PROCEDURE wallCollideCheck (walls)
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

FOR wall IN walls

IF checkCollision (wall) = True THEN

 collided = True

ENDIF

ENDFOR

ENDPROCEDURE

private PROCEDURE playerCollideCheck (player)

IF checkCollision (player) = True THEN

 collided = True

ENDIF

ENDPROCEDURE

ENDCLASS

SoundProjectile class:

CLASS SoundProjectile (INHERITS: Projectile)

private wallNum

public PROCEDURE constructor ()

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
wallNum = 0  
ENDPROCEDURE  
  
private PROCEDURE wallCollideCheck (walls, player)  
    FOR wall IN walls  
        IF checkCollision (wall) = True THEN  
            incrementWalls ()  
            IF checkCollision (player) = True THEN  
                setCollided (True)  
            ENDIF  
        ENDIF  
    ENDFOR  
ENDPROCEDURE  
  
private PROCEDURE incrementWalls ()  
    wallNum = wallNum + 1  
ENDPROCEDURE  
  
public FUNCTION getWallNum ()  
    RETURN wallNum  
ENDFUNCTION
```

ENDCLASS

Map class:

CLASS Map ()

 private height
 private width
 private floorCoords
 private wallCoords
 private doorCoords
 private map

public PROCEDURE constructor (mapHeight, mapWidth, floorDict, wallDict, doorCoord)

 height = mapHeight
 width = mapWidth
 floorCoords = floorDict
 wallCoords = wallDict
 doorCoords = doorCoord
 map = []

ENDPROCEDURE

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
public PROCEDURE createMap ()  
    widthCounter = 0  
    heightCounter = 0  
    WHILE heightCounter <= height  
        add empty list to map  
        WHILE widthCounter <= width  
            add place holder value to the map  
            widthCounter = widthCounter + 1  
        ENDWHILE  
        widthCounter = 0  
        heightCounter = heightCounter + 1  
    ENDWHILE  
    addWalls ()  
    addFloors ()  
    addDoor ()  
ENDPROCEDURE
```

```
public PROCEDURE addWalls ()  
    FOR type IN walls  
        FOR wall IN walls[type]  
            wall = create wall object
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

add wall object to corresponding place in map

ENDFOR

ENDFOR

ENDPROCEDURE

public PROCEDURE addFloors ()

FOR type IN floors

FOR floor IN floors[type]

floor = create floor object

add floor object to corresponding place in map

ENDFOR

ENDFOR

ENDPROCEDURE

public PROCEDURE addDoor ()

door = create door object

add door to corresponding place in map

ENDPROCEDURE

public FUNCTION getObject (coords)

RETURN map[coords[0]][coords[1]]

ENDFUNCTION

```
public FUNCTION getWalls (type)
    walls = []
    IF type == None THEN
        FOR wallType IN wallCoords
            FOR wallCoord IN wallCoords[wallType]
                wall = getObject (wallCoord)
                add wall to walls
            ENDFOR
        ENDFOR
    ELSE
        wallTypeCoords = wallCoords[type]
        FOR wallCoord IN wallTypeCoords
            wall = getObject (wallCoord)
            add wall to walls
        ENDFOR
    ENDIF
ENDFUNCTION
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
public FUNCTION getFloors (type)
floors = []
IF type == None THEN
    FOR floorType IN floorCoords
        FOR floorCoord IN floorCoords[floorType]
            floor = getObject (floorCoord)
            add floor to floors
        ENDFOR
    ENDFOR
ELSE
    floorTypeCoords = floorCoords[type]
    FOR floorCoord IN floorTypeCoords
        floor = getObject (floorCoord)
        add floor to floors
    ENDFOR
ENDIF
ENDFUNCTION
```

```
public FUNCTION getDoor ()
RETURN getObject (doorCoords)
ENDFUNCTION
```

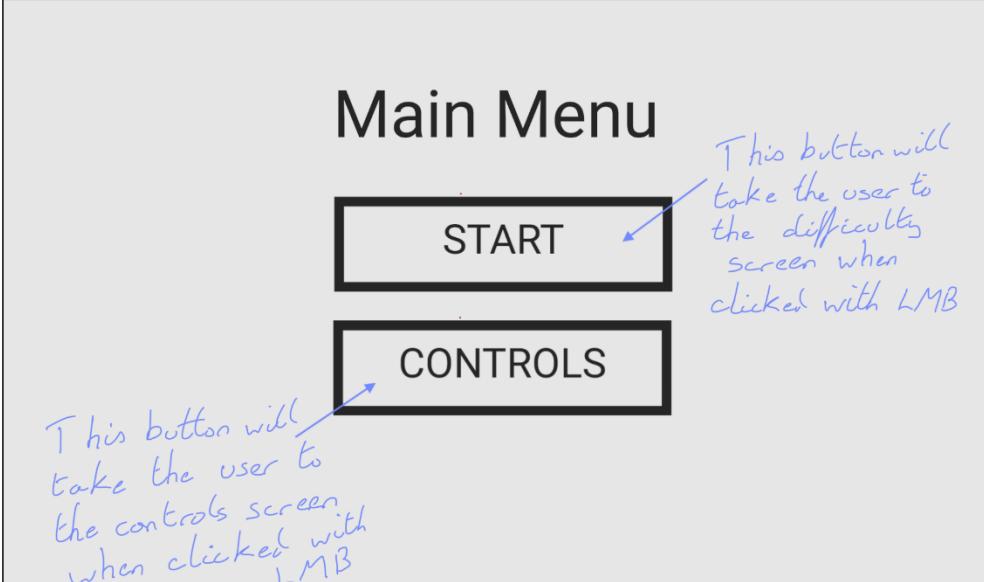
Name: Muqtasid Zayyan Dar

Project title: Manhunt

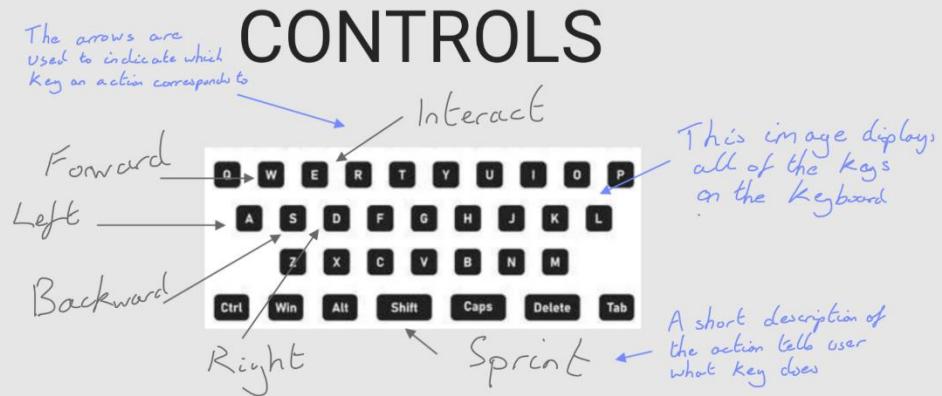
```
public FUNCTION getAllObjects ()  
    allObjects = []  
    FOR row IN map  
        FOR object IN row  
            add object to allObjects  
        ENDFOR  
    ENDFOR  
    RETURN allObjects  
ENDFUNCTION  
  
public FUNCTION getMap ()  
    RETURN map  
ENDFUNCTION  
ENDCLASS
```

Usability:

In this section I am going to be creating a drawing of how my program is going to work from the users point of view. The drawings are not to scale.

Ref	Drawing	Justification
U1	 <p>The drawing shows a "Main Menu" screen. It features two large rectangular buttons: "START" at the top and "CONTROLS" below it. Handwritten blue annotations provide details: an arrow points to the "START" button with the text "This button will take the user to the difficulty screen when clicked with LMB"; another arrow points to the "CONTROLS" button with the text "This button will take the user to the controls screen when clicked with LMB".</p>	(Annotations in blue) This is the main menu screen which will allow the user to navigate through the different screens before the game begins. When the user clicks on the start button with the left mouse button (LMB) then it will take the user to the difficulty screen (U3) which will let the user choose a difficulty for the game. If the user clicks on the controls button with the left mouse button then it will take the user to the controls screen (U2) which will let the user see the controls for the game..

U2



(Annotations in blue)

This is the controls screen, this is needed so that the user is able to learn the controls for the game if they have not played the game before. There will be an image with the keys on the keyboard in the center and arrows pointing at the keys which will tell the player what action links to which key. This way the player does not have to guess what key does what when they are playing the game and they can quickly learn the controls.

U3

Each of these buttons will result in a different difficulty for the game when clicked

DIFFICULTY

EASY

NORMAL

HARD

INSANE

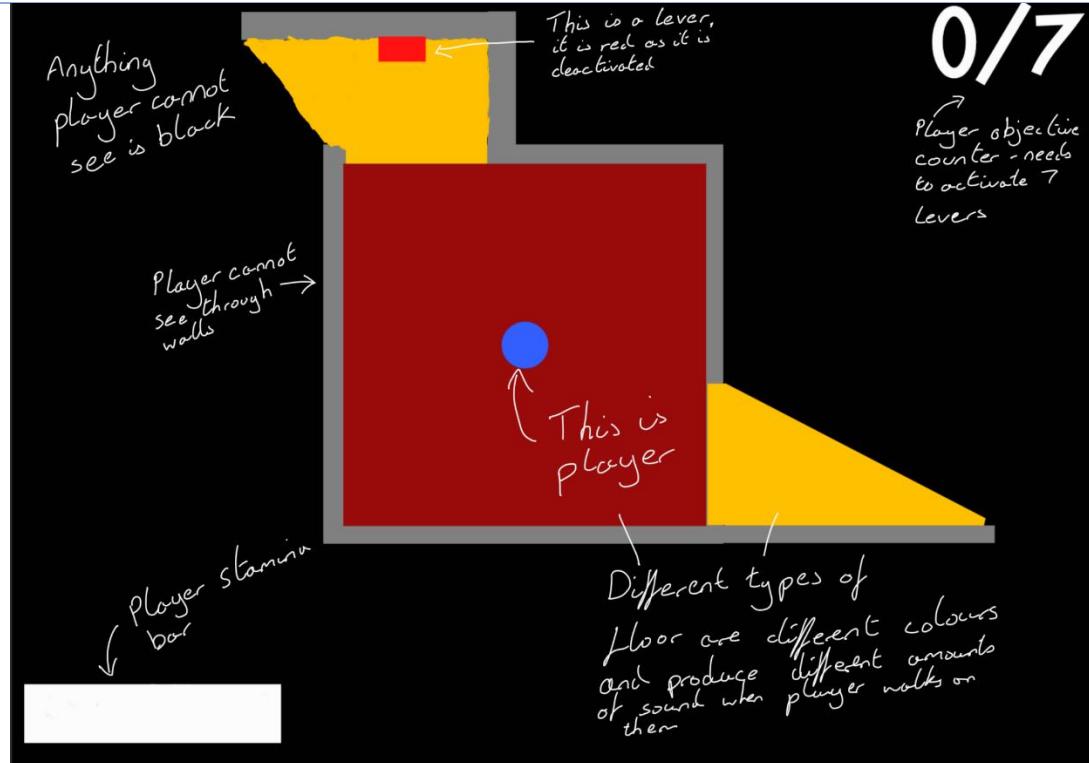
NIGHTMARE

Hardest difficulty is in red to give connotations of danger and help it stand out

(Annotations in blue)

This is the difficulty screen which is displayed after the user clicks on the start button. This screen is needed so that the user can choose a difficulty for the game before they start. Each button will change the mechanics in the game slightly, for example, the harder the difficulty, the more levers the player will have to activate before they can escape or maybe on easier difficulties, the hunter moves slower or is less likely to hear the player.

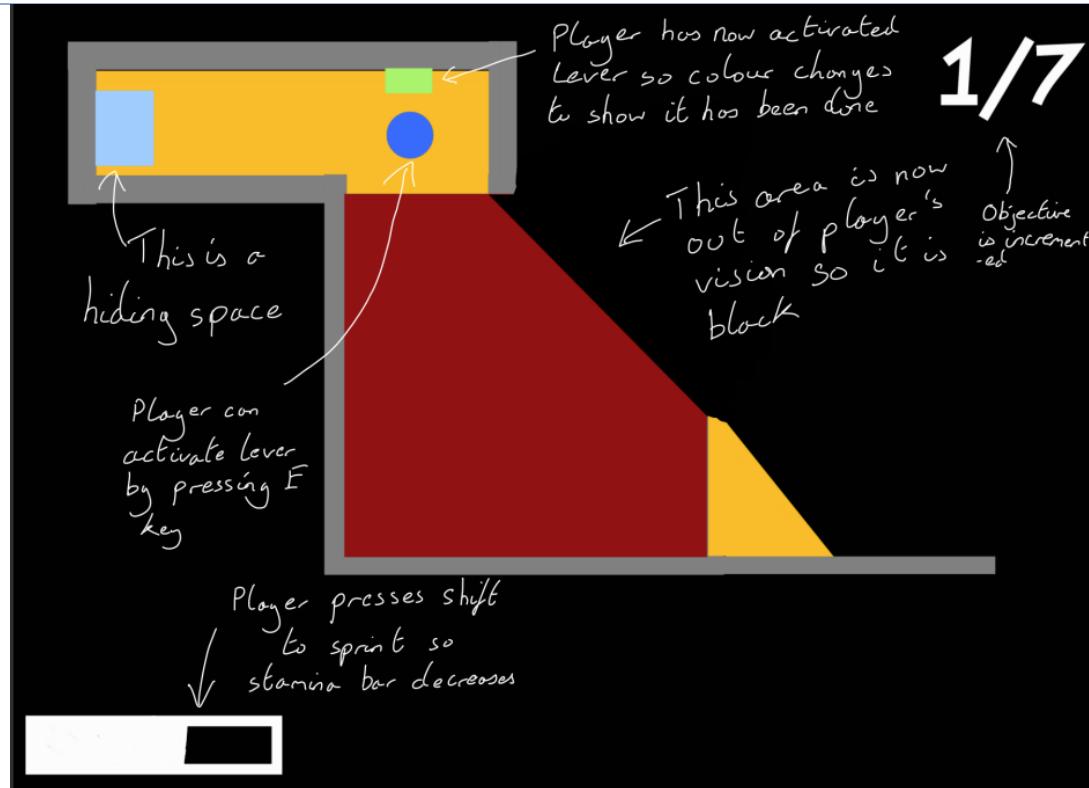
U4



(Annotations in white)

This is what the game will look like when the player first spawns in the game after the normal button is pressed in the difficulty screen (U4). Anything out of the player's field of view – determined by projectiles – will not be visible and will be black. However, anything which is visible will be displayed on the screen. The stamina bar will be kept at the bottom left of the screen so that it is easy for the user to keep an eye on how much stamina they have left while they are in the game. There is also an objective counter in the top right of the screen so that it is easy for the player to keep an eye on how close they are to being able to escape. The levers in the game will appear red when deactivated as red is always associated with off. The floors will be different colours depending on the type of floor and the player will have a simple design which is different compared to the surroundings so that the user can tell that that is their character.

U5

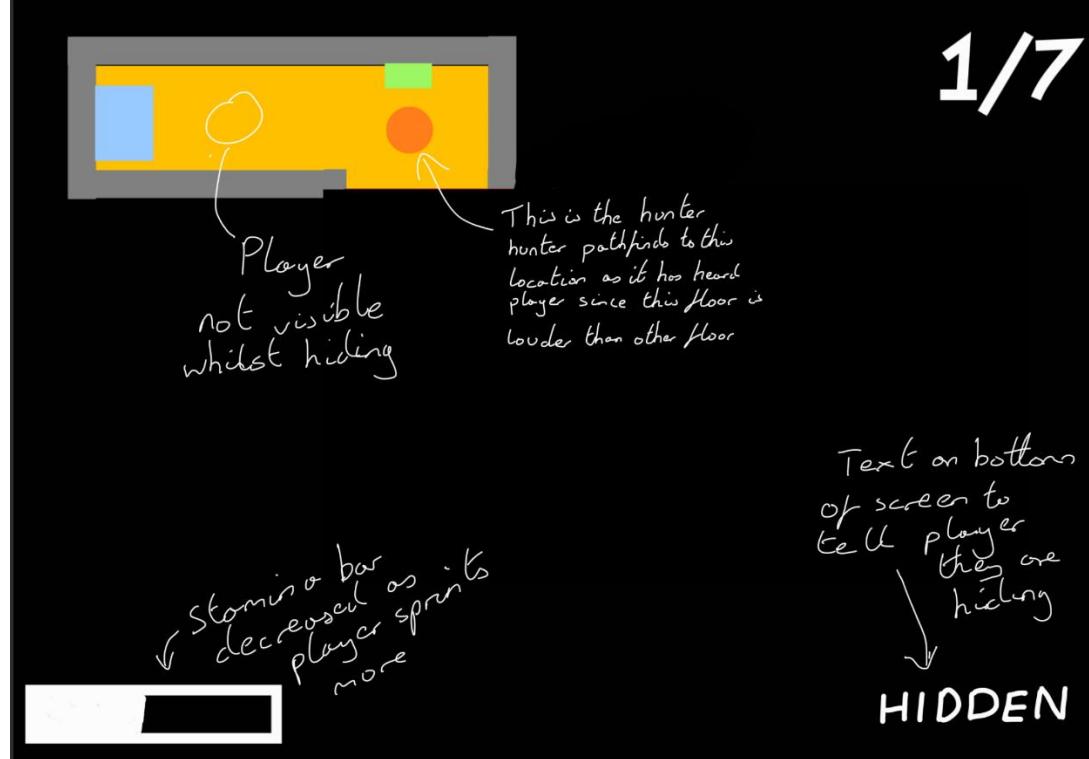


(Annotations in white)

This is an image slightly after the previous one (U4) where the user has interacted in 3 ways: the user has moved their character across the screen using the movement button while they were pressing the Shift button which makes the character move faster across the screen and also causes the stamina bar to start decreasing. The player has also pressed the E key and interacted with the lever since they are close enough causing it to change colours from red to green and increment the objective counter by 1. The stamina bar decreases so that the player has an idea of how much sprint time they have left. The lever changes colour so that the player knows that the lever has been activated, moreover, the objective counter is incremented so that the player can confirm they have activated the lever and they are able to see how close they are to beating the game. As the player has moved to a different location, the parts of the map which are visible have changed so since the player cannot see through walls, they cannot see the top right corner of the room which was visible before as a wall is covering it. Finally, the hiding space is given a completely different colour to all other objects in the environment so that the player can differentiate between the hiding space and the other objects.

U6

1/7



(Annotations in white)

This is a drawing which is depicted as what happens after the above drawing (U5), the yellow-brown floor is louder than the red one so in this situation the hunter has heard the player and has used pathfinding in order to get to the location which the player was in when the player was creating the sound. The hunter is an orange circle so that the user can quickly recognise it as being like the player as it is the same shape and size but it is a different colour so that the user is able to differentiate between themselves and the hunter. In this situation the player is also hiding in the hiding space and are not visible on the screen, however, the player should still be able to see their surroundings as if they are standing in the location of the hiding space. When the player is hidden, text on the bottom right of the screen will be used to inform the player that they are hidden so that they know when they are hiding. As the player pressed shift in this situation while moving, the stamina bar has decreased some more.

U7

1/7



Player not hiding anymore so is visible and 'HIDDEN' text not displayed

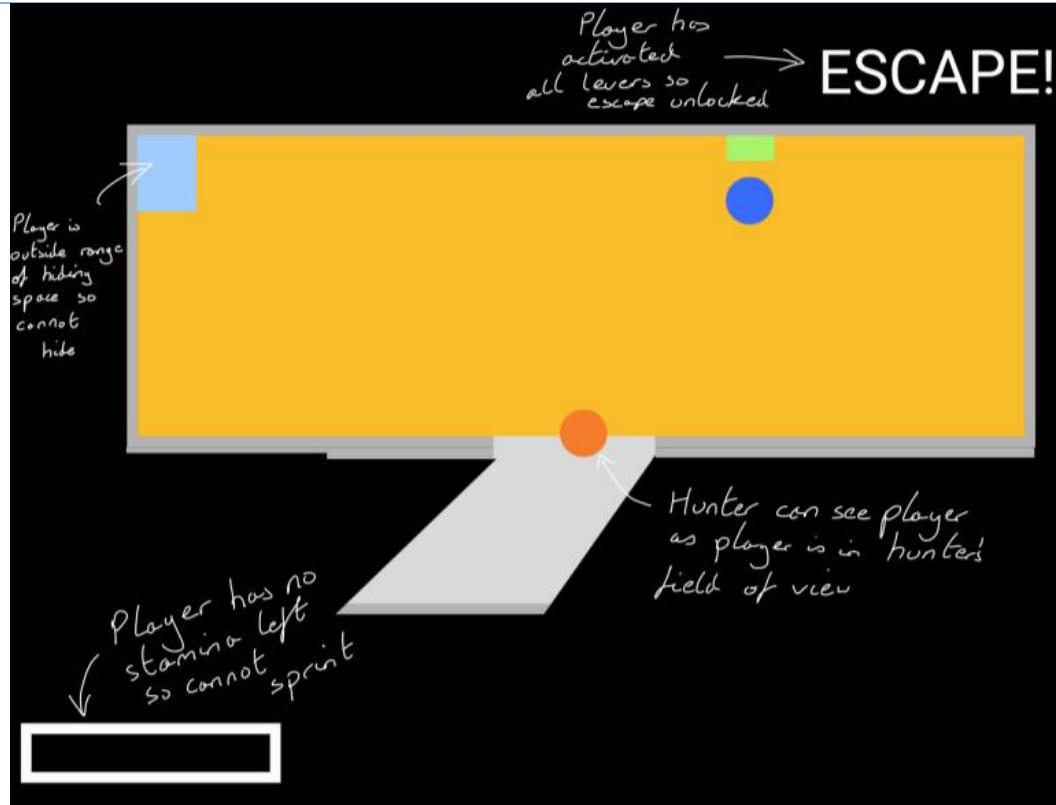
Hunter cannot see player and travels on random path to search for player

Player not moving so stamina replenishes

(Annotations in white)

This is a drawing which happens after the above drawing (U6). As the hunter could not previously see the player in its field of view and has not found the player, it will start randomly pathfinding to a different location in the map. As the hunter's field of view is in the same direction as its movement, it cannot see the player in this situation and when the player interacts with the hiding space then no sound is made so the hunter also doesn't hear the player in this situation. The player is visible on the screen again and the text on the bottom right of the screen is not displayed anymore as the player has left the hiding spot. Furthermore, as the player was not moving while being in the hiding spot, the player's stamina replenishes, however, it doesn't instantly fill up, it gradually increases over time to a maximum.

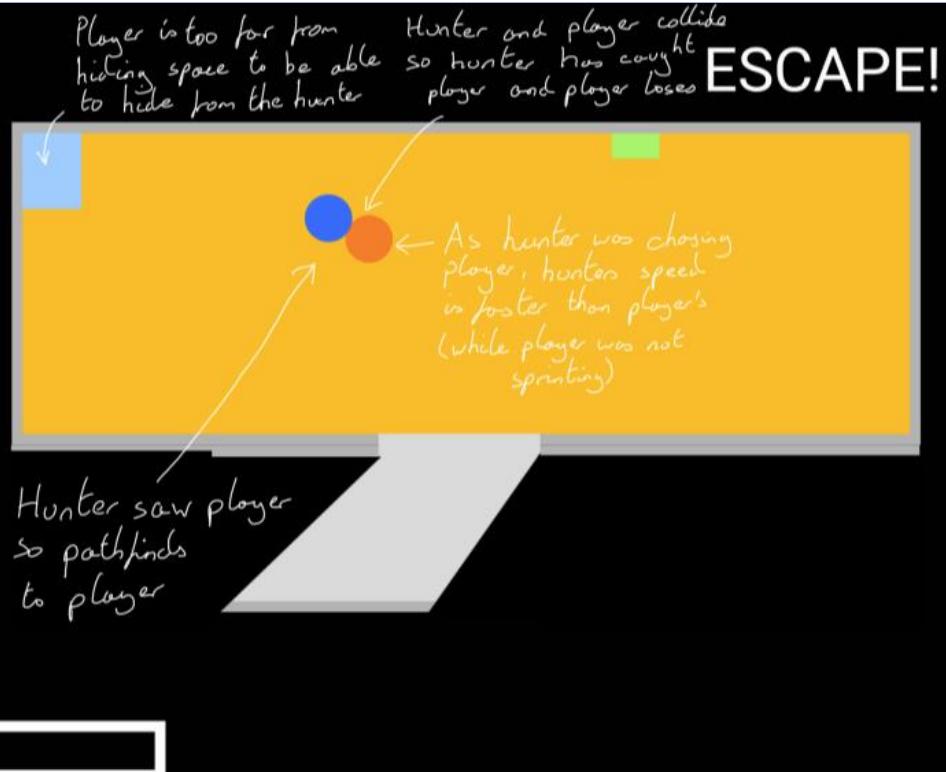
U8



(Annotations in white)

This is a different situation in the game where the hunter has heard the player and has used pathfinding to reach that location in which the player currently is. As the hunter is moving in the direction of the player and the field of view of the hunter is in the direction it is moving in, the hunter would see the player and would start chasing them which is represented in the next drawing below (U9). Furthermore, the player has no more stamina left as they have used it all so the player would not be able to move faster in this situation even if they pressed the shift button. The player is also too far away from the hiding space to interact with it and hide. However, the player has activated all of the levers so the objective counter has been changed from a counter to text which tells the player to escape, this is useful for the player as it tells them what their new objective is so that they can win the game rather than just waiting for them to figure it out.

U9



(Annotations in white)

This is a continuation of the drawing above (U8). As the hunter had seen the player in the previous situation, the hunter would pathfind to the location of the player while the player is still in the hunter's vision which happens in this situation. However, since the hunter is now chasing the player, the speed at which the hunter is moving at is higher than the player's regular movement speed which is why the hunter is able to catch the player in this situation (player still cannot sprint as no stamina left). As the hiding space is still too far away from the player for the player to be able to interact with it, the player is not able to hide from the hunter. In this situation the hunter is also close enough to the player that they can catch the player so the player would lose and the losing screen would be displayed (U10)

U10

YOU HAVE BEEN CAUGHT !

TIME SURVIVED:

1: 37

Time taken for the player
is displayed so the user knows
whether they have improved

MAIN MENU

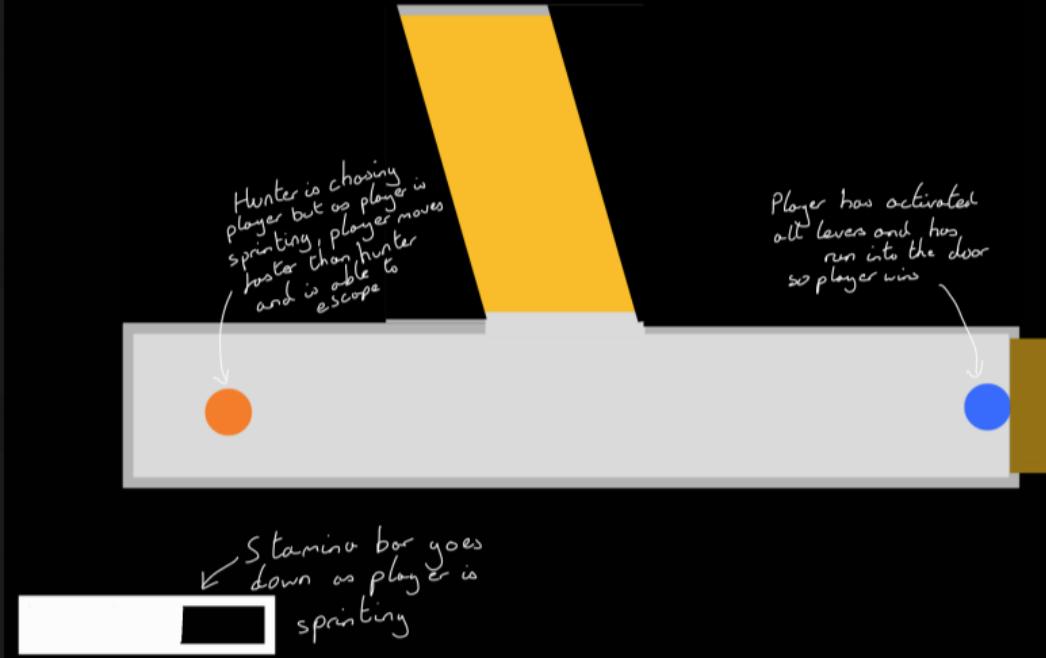
Main Menu button returns player
to menu so they can play again

(Annotations in blue)

This screen is needed so that the player knows that they have lost the game. The text at the top tells the user that they have been caught by the hunter which means that they have lost the game. The time which they have not been caught by the hunter is displayed below that, this is helpful for the user as it allows them to be able to see whether or not they have improved from previous times where they have played. Finally, the main menu button is in the middle of the screen so that the user is able to go back to the main menu and play the game if they want.

U11

ESCAPE !



(Annotations in white)

This is another situation similar to U9 as the player has activated all of the levers in the map and the hunter is chasing the player. However, in this situation the player still has some stamina and is sprinting so they are faster than the hunter while the hunter is chasing the player. This is useful for the user as it means that they have a way to escape the hunter if they do not have a hiding space close to them. Furthermore, the player has run into the escape door which is the brown rectangle. This means that the player has won the game as they have now escaped through the escape door and will be displayed the win screen.

U12

YOU HAVE ESCAPED !

TIME SURVIVED:

3:14

← Time taken displayed
so user knows whether
they have improved or
not

MAIN MENU

Tells player
they have
won game

Main Menu button so
user can play again

(Annotations in blue)

This is the screen which is displayed after the player has won, it is similar to the losing screen, however, the text at the top is different. This is so that the player knows that they have not lost the game since different text shows up at the top. Similar to the losing screen the time is displayed below the text and the main menu button is in the center of the screen for the same reasons.

Approach to testing:

In Development Testing:

For each of these main parts of the program I will be going through how I will be testing each of them during development.

4 Screens:

As I develop this part of the program, to test it, firstly I will need to check if the screen will display when the program is run. Then after I code in the buttons for the screens, I will need to test those in 2 ways:

- I need to test if the button works when it is clicked by the left mouse button. For this the valid inputs would be a left mouse click where the cursor is over the top of the button, an invalid input would be any left click that is not above the button.
- I also need to test whether it takes the user to the correct place after it has been clicked
- For the game screen I will need to check if the map is being displayed correctly with the correct objects in the correct places and also that the walls with hiding spaces and levers are distinct so that the user can easily tell the difference between normal walls and other interactive walls.

Hunter:

While developing the hunter I will need to test:

- The hunter's FOV:
 - To test this I will have the player in various positions around the hunter, some of which the hunter should be able to see the player in, for example, right in front of the hunter with no walls and in other situations where the hunter should not be able to see the player such as when the player is behind a wall or when the player is outside the hunter's FOV
- The hunter's pathfinding:
 - To test this I will have to test the different aspects of the hunter's pathfinding:
 - Random paths – I will need to test this by seeing if the hunter just randomly travels around the map if the hunter has not seen or heard the player
 - Paths to sound – The player will have to create a sound or I may have to artificially create a sound to check if the hunter is able to pathfind correctly to the origin of the sound
 - Paths when player is in sight – to do this I will need to test times when the player is within the hunter's FOV and see if the hunter moves directly towards the player.

- Paths when the hunter has lost sight of the player – to do this I will need to first move the player into the hunter's FOV and then either hide in a hiding space or run to somewhere the hunter cannot see the player. Then I will need to check if the hunter predicts where the player went and then travels in that direction. After this path the hunter should also revert to random pathfinding
- The hunter's Listening:
 - To test this I will have to create a variety of sounds on different floor surfaces at different distances from the hunter and then see whether the hunter is able to pathfind to the location of the sound. I will have to do the same test multiple times for each distance and floors to test if the probabilities of the hunter hearing the player are as they should be
- Collisions:
 - I will have to check to make sure that the hunter cannot move through walls and that the hunter is not able to move through the player.

Player:

While developing the player I will need to test the:

- Movement:
 - The player should be able to move forwards, left, backwards and right using the W, A, S and D buttons respectively
 - I will need to make sure that when the player is pressing the sprint button ('Shift') and is moving in a direction that they are moving at a faster speed than when they are just pressing one of the movement buttons.
 - I will need to make sure that when the
- Interactions:
 - When the player presses the E button it should allow them to interact with levers and hiding spaces provided that the player is within the activation area of the lever or the hidingspace:
 - If the player has interacted with the lever then it should increment the lever counter by 1 and set the lever as activated.
 - If the player has interacted with the hidingspace then the player should be invisible to the hunter and should appear as if they have gone into the hiding space and are not on the screen anymore.
 - If the player is not within the activation area of the lever or the hidingspace then the lever or hiding space should not carry out its respective task
- Collisions:
 - if the player walks directly into a wall or the hunter, they should not be able to walk through it

Map:

- As I develop this part of the program, to test it I will first need to test each of the object classes that I have coded starting with the main parent object class which all the other objects will inherit attributes and methods from.
- To test this part of the program I will first check that all of the getter and setter methods return the right things and change the correct things.
- Next, After coding each of the object classes: wall, floor, door, lever and hidingspace, I will test each of these.
 - For the wall, I will test that when a valid type of wall is entered that the type of wall created is whatever type was entered. I will also enter an invalid type of wall to check whether the class accepts it.
 - For the floor class, I will do a similar thing as there are different types of floors.
 - For the door class, to test it I will check that the door state is only set as opened when the number of activated levers is the same as the required number of levers.
 - I will test the lever class by activating the lever and deactivating the lever and then checking the state of the lever to see if it has been activated or not, I will also test it later on to check if the player is able to activate it while they are outside the activation area and whether they are able to activate it while they are within the activation area.
 - For the hidingSpace class I will do a similar thing with the lever where I will see if it carries out the relevant actions on the player once I have also coded some part of the player.
 - I will also test whether the hunter is able to see the player after the player has activated the hiding spot once the other parts have been coded.
 - I will also need to check that the player's state reverts once the player has left the hiding spot.
- After this I need to code the map class itself, which generates the game map, to test this I need to check:
 - If the size of the map generated is the same as what has been entered as the size
 - If the objects are added to the map list in the correct location
 - If the right object is returned when trying to get an object via the coordinates.
 - The correct objects are returned when searching for a type of object

Physics:

- To test the physics, while I am coding it I will need to run the program and then move the player and the hunter around to see if they are able to walk through walls or each other. I will have to check collisions from all angles of the wall to ensure that the player and hunter cannot move through them.
- I also need to ensure that when the player and the hunter walk into each other, it registers as a collision and leads to the round ending.

Name: Muqtasid Zayyan Dar

Project title: Manhunt

- I also need to check when 2 buttons are pressed that if they are the opposite the player doesn't move and if the buttons are at right angles to one another that Pythagoras is used to calculate the diagonal movement.
- Another thing that I need to test is that only the WASD buttons should make the player move, any other button on the keyboard should not affect the player's movement.
- Each button should make the player move at the same magnitude of speed but in different directions.
- When the player is sprinting then the player's speed should be increased and it should move faster on the screen.
-

Round Ends:

- As I develop this part of the program, for the player to win, they will need to run through the open door. To test this I will make the player run into the door while it is closed and check that nothing happens and the game continues.
- I will also check once the door is open and the player collides with the door that it should end the game and display the winning screen and return the player to the main menu.
- After this I will need to test that the round ending screen is displayed after the player wins the game and that it is only displayed after the player moves through the open door.
- Another thing I will need to test is that after the hunter has 'caught' the player that it should take the user to a different round ending screen and this should only happen when the player has been caught by the hunter.

Post Development Testing:

Test No	Ref	Pre-Reqs	Valid/ Invalid/ Borderline	Input/Calc	Expected Outcome
T1	i1 & O1	Player isn't hiding	Valid	'W'	Character faces North on screen and moves in that direction
T2	i2 & O2	Player isn't hiding	Valid	'S'	Character faces South on screen and moves in that direction
T3	i3 & O3	Player isn't hiding	Valid	'A'	Character faces West on screen and moves in that direction
T4	i4 & O4	Player isn't hiding	Valid	'D'	Character faces East on screen and moves in that direction

Name: Muqtasid Zayyan Dar

Project title: Manhunt

T5	i5 & O8	Player isn't hiding Player have enough sprint energy	Valid	Any directional button and 'Shift'	Character moves faster in whatever direction of that key and the sprint energy decreases
T6	i5	Player has no sprint energy left	Invalid	Any directional button and 'Shift'	Character should not sprint and should only move in whatever direction at normal speed
T7	i6, O7, P13 & P15	Player is within activation radius of a hiding spot or a lever	Valid	'E'	Character should carry out whatever action depending on what they are standing close to
T8	i6, O7, P13 & P15	Player is outside activation radius of a hiding spot or a lever	Invalid	'E'	Character should not do anything
T9	i6, P13 & P15	Player is on the edge of the activation radius of a hiding spot or lever	Borderline valid	'E'	Character should carry out whatever action depending on what they are standing close to
T10	i7	Player is in menu	Valid	Left clicks on start button with mouse	Should take the user to the difficulty screen
T11	i7	Player is in menu	Invalid	Right clicks on start button with mouse	Should not do anything

T12	P1	Player is in menu	Valid	Left clicks on the options button with mouse	Should take user to options menu with list of controls
T13	P2	Player is in difficulty menu	Valid	Left clicks on either normal mode or nightmare mode	Should take user to a separate screen which has tips for the player and ends the screen once the map has loaded in
T14	P4 & P5	Player is in the game	Valid	Player moves into the hunter's FOV	The hunter should see the player and start chasing them at a faster speed and moving directly towards the player
T15	P4& P5	Player is in the game Player is outside hunter's FOV and hearing range	Invalid		The hunter should not start chasing the player
T16	P6 & P23	Player has been seen by the hunter	Valid	'Shift'	When the player starts sprinting, it should seem that the hunter is being left behind
T17	P7 & Ae1	Player has been seen by the hunter	Valid		There should be some alert which tells the player that they have been seen by the hunter
T18	P8	Player is moving	Valid		There should be a sound circle originating from the player as they are moving
T19	P8	Player is sprinting	Valid	'Shift'	There should be a bigger sound circle compared to before
T20	P10	Player's sound circle and hunter's sound circle overlap	Valid	%overlap divided by number of walls between the hunter and	Hunter should have a random chance of 0.3 to hear the player

Name: Muqtasid Zayyan Dar

Project title: Manhunt

				the player e.g. 30/1	
T21	P10	Player's sound circle and hunter's circle are on the border of one another	Borderline Invalid		The hunter should not have any chance of hearing the player as there is no overlap
T22	P11	Player is not hiding Player just left hunter's FOV	Valid		The hunter should generate a random predicted path to follow and at the end of the predicted path the hunter should return to its usual random paths
T23	P14	Player is in hunter's FOV	Valid	Player presses 'E' to hide	Hunter catches player even while the player is in the hiding spot
T24	P14	Player was in hunter's FOV	Invalid	Player presses 'E' to hide	Hunter should start searching for the player and not catch the player in the hiding space
T25	P15 & O10	Player is within activation radius of lever	Valid	'E'	The lever counter on the HUD should increment by one and the light next to the lever should turn green
T26	P17 & O9	Player has activated all of the levers	Valid		The escape door should be opened and there should be an alert informing that it has opened
T27	P17	Player hasn't activated all of the levers	Invalid		The escape door should remain closed
T28	P18	Player has activated all of the levers	Valid	Player walks through the escape door	An end screen should show up informing the player they have escaped before returning the player to the menu
T29	P19	User is on the difficulty screen	Valid	Left click on either normal or nightmare button	The relevant map and hunter should be loaded in

Name: Muqtasid Zayyan Dar

Project title: Manhunt

T30	P21	Wall(s) between the player and the hunter	Invalid		Hunter should just continue as normal
T31	P22 & O6	Player is within the catch radius	Valid		Hunter should catch the player and losing screen should be displayed
T32	P22 & O6	Player is outside the catch radius	Invalid		Hunter should not catch the player
T33	P24	Player is moving	Valid	'Shift'	Stamina bar on HUD in bottom right of screen should start decreasing
T34	P24	Player isn't moving	Invalid	'Shift'	Stamina bar shouldn't decrease
T35	P25	Player isn't sprinting	Valid	Not Shift	Stamina bar should start replenishing
T36	O5	Nothing in front of the player's character	Valid	'W'	Character should move
T37	O5	Wall in front of player's character	Invalid	'W'	Character shouldn't move
T38	O11	Player is enclosed by walls around them	Valid		The player should be able to see everything around them up to the walls
T39	O12	Player stamina bar is low/empty	Valid		An alert informing the player to wait for the stamina bar to replenish should pop up
T40					

Development

I will be using visual studio code as my source code editor for my project and I will be separating my project into different versions during development, each version will have specific objectives which I will aim to code. Small changes will be represented by subsections of a version, for major changes, a new version will be stated with added requirements.

Version 1:

In this version I am going to code the screens: main menu, options, difficulty/mode and also part of the game screen. I will also create 2 different files for the map and load them in. The user should be able to use the buttons on each of the screens and those buttons should take the user to the corresponding screen. When the user presses one of the buttons in the difficulty/mode screen, the corresponding map should be loaded in – if the normal button is pressed then the file containing the normal map should be read and that map is loaded in and the same thing for the nightmare button.

Version 1.1:

In this version of the game I am going to use a temporary screen and code a buttons class in which will display buttons on the screen which the user can click and will do different things

```
import pygame

#Initialising the pygame module
pygame.init()

#Creating a screen
screenHeight = 640
screenWidth = 960
screen = pygame.display.set_mode((screenWidth, screenHeight))

#Setting a title
pygame.display.set_caption("Manhunt")

#Setting the icon
gameIcon = pygame.image.load('Manhunt.png')
pygame.display.set_icon(gameIcon)
```

First I initialized the Pygame library. Then I set the height and width of the screen and also set the caption of the window to be 'Manhunt'. Then I also loaded in an image and then set that as the icon in the corner.

```
#Procedure to display image
def displayImg(imgName, x, y):
    #Loads the image
    img = pygame.image.load(imgName)
    screen.blit(img, (x, y))

#Procedure to display text
def displayText(textName, x, y, size, colour = (0,0,0), font = None):
    #Sets font(I/A) and size of the text
    textFont = pygame.font.Font(font, size)
    #Renders the text with anti aliasing(Boolean) and colour (Tuple)
    text = textFont.render(textName, True, colour)
    screen.blit(text, (x, y))
```

I created some procedures so that I can call them anytime throughout the code as they are more efficient and reduce redundancy within the code. The first procedure displays an image, it is passed a string with the name of the image in the file e.g. ‘Manhunt.png’ and then the x and y coordinate of the pixels where it will be displayed. Within the procedure, the image is loaded and then displayed on the screen with those coordinates. The second procedure for displaying text is similar as it is passed the name and the coordinates, however, it is also passed the size of the text (in pixels), the colour (tuple) and the font if needed. Then the font and size are set and the text is rendered with the font, size, anti-aliasing and also color. Finally, the text is displayed with those coordinates.

```
#Loop for game screen
running = True
while running:
    #event handler
    for event in pygame.event.get():
        #Checks through all of the events that are happening in the window
        #If user presses cross button window closed
        if event.type == pygame.QUIT:
            running = False

    #Changing background colour
    screen.fill((75,75,75))
    displayText('Manhunt', 230, 80, 160)
    displayImg('Manhunt.png', 32, 32)
    displayImg('rectangleStart.png', 350, 160)
    pygame.display.update()
```

I created a while loop which keeps the game running. The nested for loop checks all the events that are happening and then the if statement checks if the event is the user pressing the cross button on the

window. If the user does then running is set to false and the program ends. Within the while loop, the other lines adjust the graphics for the screen by displaying images or also changing the background colour.

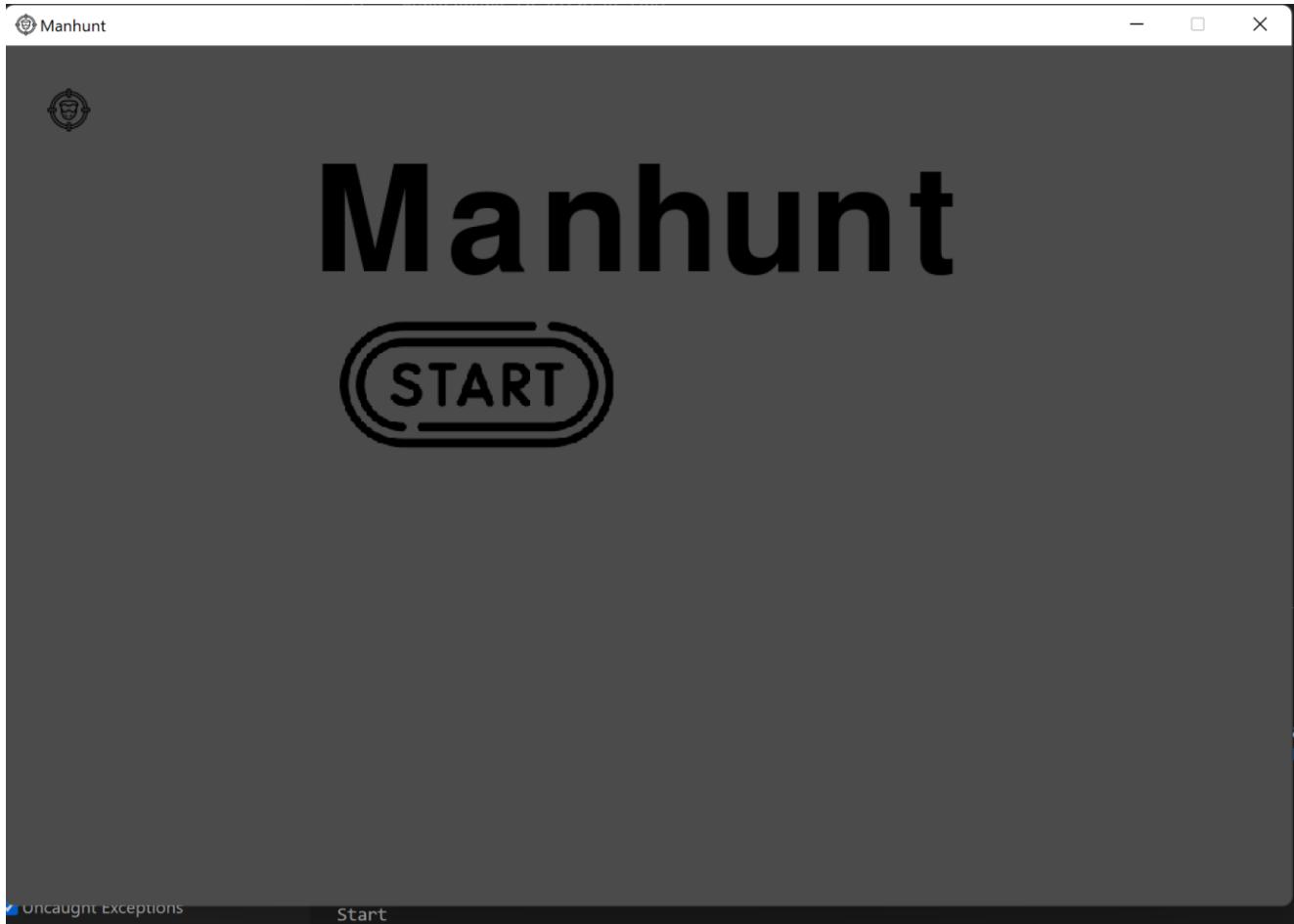
```
class Button():
    def __init__(self, img, x, y, scale):
        #Loads the image
        self.img = pygame.image.load(img).convert_alpha()
        #Gets the width and height of the img in pixels
        width = self.img.get_width()
        height = self.img.get_height()
        print(width)
        print(height)
        #Transforms the image using a scale
        self.transformedImg = pygame.transform.scale(self.img, (int(width * scale), int(height * scale)))
        #Gets the rectangular area of the image
        self.rect = self.transformedImg.get_rect()
        #Is the top left of the image
        self.rect.topleft = (x, y)
        self.clicked = False
```

This is the button class which takes an images, the x coordinate and the y coordinate and a scale and then displays the image based on those attributes. First the image is loaded in and then built in pygame methods are used to then get the height and width of the image. After this, the scale can be used to then change the size of the image by using the transform method. Both the height and width are multiplied by the scale to change the size of it. Then the .get_rect() method is used on the transformed image which gets the rectangular area of the image on the screen. the final two are the top left of where the image would be displayed on the screen and self.clicked is used later on in a method.

```
def draw(self):
    #draws image on the screen
    screen.blit(self.transformedImg, (self.rect.x, self.rect.y))

def clickCheck(self):
    self.draw()
    hasClicked = False
    pos = pygame.mouse.get_pos()
    #print(pos)
    #Is mouse cursor colliding with the rectangle of image
    if self.rect.collidepoint(pos):
        #Checks if the mouse has been pressed and has already been pressed before
        if pygame.mouse.get_pressed()[0] == 1 and self.clicked == False:
            #Set to true so that button is only registered once with one mouse click
            self.clicked = True
            hasClicked = True
        #If the user is not pressing the mouse button it is reset so that the user
        #can use it again
        if pygame.mouse.get_pressed()[0] == 0:
            self.clicked = False
    #Returns whether the mouse has been clicked or not
    return hasClicked
```

In the first method, the transformed image is displayed on the screen depending on the coordinates of the top left of the image. The second method uses the first method to draw the image on the screen and the positions of the mouse is assigned to the variable. The first statement checks if the pointer is above the image and the nested if statement checks whether the user has clicked on the left mouse button and if the variable self.clicked is false. If both of those scenarios are fulfilled then the variable self.clicked and hasClicked are set to true. self.clicked is then set to false again and hasClicked is returned which informs whether the button has been clicked (True) or not (False). Also the variable self.clicked is used so that each button press from the user is only registered as one button press. It is set to false again later so that the button can be reset and used again.



This is the test, the images are displayed in the relevant places based on the coordinate of the top left pixel. In this situation, the start image on the screen is the button, and when it is clicked with the left mouse button by the user then it prints start, which is displayed in the image above.

Review:

In this iteration of the program, I learned how to use the pygame library in order to display text and images on the screen. I also learned how to run a game loop in order to be able to run my program until the user chooses to quit the program. By creating and displaying a button on the screen, I learnt how to implement my ability to display images on the screen and use it for a different situation. Furthermore, I learnt some of the functions of the pygame library which can help with detecting where the user's cursor is on the screen and when the cursor is in the relevant place.

Specs completed/removed:

- I7 – User can successfully navigate start screen using their mouse

Specs ongoing:

- None

Specs started:

- P1 – Other screens have not been implemented yet.

Version 1.2:

In this version I am going to code the screen class which should be able to take the user to different screens and also display images, texts and the buttons on the screen. The buttons should work when clicked on.

```
#Screen class
class Screen():
    def __init__(self):
        #All lists are 2 dimensional, storing what needs to be displayed
        #and also the coordinates of where they should be displayed
        self.texts = []
        self.images = []
        self.buttons = []
        #All screens will be the same size
        self.screen = pygame.display.set_mode((960, 640))
```

I decided to make a class for the different screens as I realised the code would be quite repetitive if I decided to code each screen separately. The class takes no parameters however, it has the attributes: texts, images, buttons and then screen which just creates a window with a certain size. The texts, images and buttons attributes are all 2D lists, however, their contents are slightly different.

```
#Procedure to display image
def displayImg(self):
    imageIndex = 0
    #Runs while both of the counters above haven't gone above the length each list within the list
    while imageIndex != len(self.images[0]):
        #Loads the image
        img = pygame.image.load(self.images[0][imageIndex]).convert_alpha()
        width = img.get_width()
        height = img.get_height()
        scaleImg = pygame.transform.scale(img, (int(width * self.images[2][imageIndex]), int(height * self.images[2][imageIndex])))
        self.screen.blit(scaleImg, self.images[1][imageIndex])
        imageIndex += 1
```

This is the procedure to display images. It only takes self as the parameter and a while loop is used to go through every element within the self.images list. For each image in that list, it will load the image, get the width and height of the image and then set the scale for that image. After that it will display the image on the screen. The loop ends after all images have been displayed on the screen. The way the self.images list works is that the name of the image is within the first list, then the coordinates of where the image should go is within the second list and finally, in the last list there is the scale of the image. The elements of each list within the list are put in order with the relevant images.

```
#Procedure to render multiple lines of text
def renderMTtexts(self, texts, sizes, colours, fonts, textCoords):
    textList = []
    counter = 0
    #Goes though each string of text in the list
    while counter != len(texts):
        #Uses the renderText method to render the text
        render = self.renderText(texts[counter], sizes[counter], colours[counter], fonts[counter])
        #Appends each render of texts to the textList for later
        textList.append(render)
        counter += 1
        #Both the list of rendered texts and the list of coordinates are
        #appended to the texts list within the class
    self.texts.append(textList)
    self.texts.append(textCoords)
```

This is the procedure to render and append multiple texts that need to show on screen to the self.texts list along with the coordinates. The procedure has parameters which are all passed as lists: texts – holds all of the text that should appear on screen; sizes – which holds the sizes of each of the text when they appear on screen; colours – which holds all of the colours for each text, fonts – holds the font for each text and textCoords – this is simply the coordinates of where the text should appear on screen. The while loop runs for as many elements there are within the first list in the 2D list which would be where all of the text that should appear on screen is. Within the while loop, the text is rendered using counter as an index which is the same for all of the lists as they are in order of the text. Then after a text is rendered, it is appended to the textList list where it is held and the counter is incremented by 1. After the while loop is finished then the textList is appended to self.texts and so is the textCoords

```
#Function to render text
def renderText(self, textName, size, colour, font):
    #Sets font(I/A) and size of the text
    textFont = pygame.font.Font(font, size)
    #Renders the text with anti aliasing(Boolean) and colour (Tuple)
    text = textFont.render(textName, True, colour).convert_alpha()
    return text
```

This function renders an individual piece of text. First, the font of the text is set and then the text is rendered and that text is returned.

```
#Procedure to display text
def displayText(self):
    textIndex = 0
    #Runs while both of the counters above haven't gone above the length each list within the list
    while textIndex != len(self.texts[0]):
        self.screen.blit(self.texts[0][textIndex], self.texts[1][textIndex])
        textIndex += 1
```

This procedure goes through self.texts using a while loop and then uses the rendered texts and the coordinates to display it on the screen and then increments the textIndex by one to display the next text.

```
#Procedure which accesses the Button class to create a button and then attributes it to this specific class
def createButton(self, id, image, x, y, scale):
    #Creates an object of type button with the relevant attributes
    button = Button(id, image, x, y, scale)
    #Adds the button to the list of buttons
    self.buttons.append(button)
```

This procedure takes the parameters id, image, x, y and scale in order to make an object of class button. This is then appended to the buttons list

```
#Function which searches and returns a button based on the ID of the button
def searchButton(self, id):
    #Goes through each of the buttons within the list
    for button in self.buttons:
        #Compares the id of the current button to the
        #one it is searching for
        if button.id == id:
            #returns the button object
            return button
```

This is the search button function which goes through each button within the list of buttons and if the id matches what the function has been passed then it is returned.

```
#Procedure which displays a new screen
def displayScreen(self):
    #Colours the screen and covers all blitted things
    self.screen.fill((150,150,150))
    self.displayImg()
    self.displayText()
```

This procedure displays a new screen which covers all of the previous text, images and buttons and then the buttons of this particular screen is then projected on top of the new screen

```
#Procedure which appends all image based things into the images list
def addImages(self, images, Coords, scale):
    self.images.append(images)
    self.images.append(Coords)
    self.images.append(scale)
```

This procedure just takes 3 lists as parameters and then each is appended to the images list

Review:

In this iteration of the program, I learnt how to implement what I learnt about displaying screens into a modular format which helped since I have to make multiple screens for the menu. I also learnt how to decompose the main problem into many smaller problems which can each then be solved by creating a method.

Specs completed/removed:

- None

Specs ongoing:

- P1

Specs started:

- P19

Version 1.3:

In this version I am going to implement the screens, and create different functions to display different screens. Each button should carry out what it is meant to do. For example, the settings button should take the user to the settings screen and display that screen.

```
#Screens form screen class
mainMenu = classes.Screen()
settings = classes.Screen()
```

These are the two objects of type Screen which are the main menu and settings

```
#Variables for mainMenu
mainMenu_texts = ['Manhunt', 'SUI']
mainMenu_textSizes = [160, 130]
mainMenu_textColours = [black, white]
mainMenu_textFonts = [None, None]
mainMenu_textCoords = [(230, 80), (230, 240)]
mainMenu_images = ['Manhunt.png']
mainMenu_imagesCoords = [(32, 32)]
mainMenu_imagescales = [1]
mainMenu.renderMTexts(mainMenu_texts, mainMenu_textSizes, mainMenu_textColours, mainMenu_textFonts, mainMenu_textCoords)
mainMenu.createButton('start', 'rectangleStart.png', 250, 150, 0.8)
mainMenu.createButton('options', 'settings.png', 5, 585, 0.1)
mainMenu.addImages(mainMenu_images, mainMenu_imagesCoords, mainMenu_imagescales)
mainMenu.setColour((150, 150, 150))
```

These are all the variables that are passed to the screen for the main menu

```
#Variables for settings class
settings_texts = ['Settings', 'SUI']
settings_textSizes = [160, 130]
settings_textColours = [black, white]
settings_textFonts = [None, None]
settings_textCoords = [(230, 80), (230, 500)]
settings_images = ['start.png']
settings_imagesCoords = [(250, 200)]
settings_imageScales = [1]
settings.renderMTexts(settings_texts, settings_textSizes, settings_textColours, settings_textFonts, settings_textCoords)
settings.createButton('start', 'rectangleStart.png', 250, 150, 0.8)
settings.createButton('options', 'settings.png', 5, 585, 0.1)
settings.addImages(settings_images, settings_imagesCoords, settings_imageScales)
settings.setColour((200, 200, 200))
```

These are all of the variables that are passed to the settings screen

```
def mainMenuScreen(mainMenu, settings):
    running = True
    while running:
        mainMenu.displayScreen()
        if mainMenu.searchButton('start').clickCheck(mainMenu.screen) == True:
            print('Start')
        if mainMenu.searchButton('options').clickCheck(mainMenu.screen) == True:
            running = settingsScreen(settings)
            print('settings')
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        pygame.display.update()
```

This function does the while loop which was shown at the start, however, this time the displayScreen procedure is used to display the mainMenu screen and then based on that different conditions can be put in place which results in different outputs. For example if the user clicks on the start button it will print start. However, if the user clicks on the options image then the settings screen will be displayed using that function

```
def settingsScreen(settings):
    running = True
    while running:
        settings.displayScreen()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False

        pygame.display.update()
```

This is the procedure so far for the settings screen, it does a similar thing to the mainMenu screen but it uses the settings object instead. When the user presses the cross in the top right corner of the screen then the function returns false so that all the screens are closed at the same time.

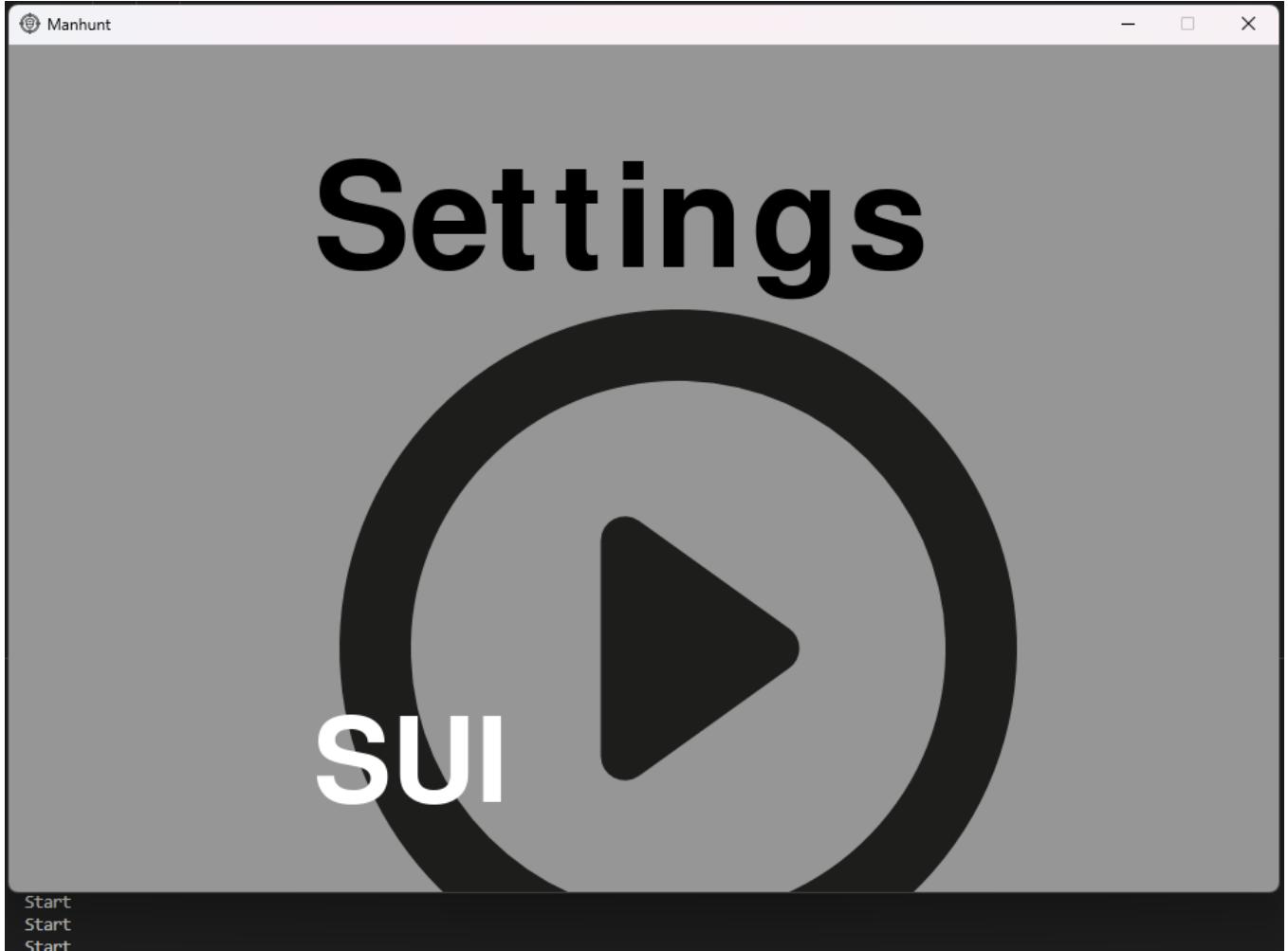
```
mainMenuScreen(mainMenu, settings)
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt



This is what is displayed on the main menu screen when the code is run and when start is pressed then start is printed as can be seen in the bottom left



When the settings button is pressed in the bottom then the new screen is displayed and the different texts and images are displayed. All the images and text used so far are for testing and aren't guaranteed to be in the final product.

```

def modeScreen(mode):
    running = True
    while running:
        mode.displayScreen()
        if mode.searchButton('return').clickCheck(mode.screen) == True:
            return True
        if mode.searchButton('normal').clickCheck(mode.screen) == True:
            print('normal')
        if mode.searchButton('nightmare').clickCheck(mode.screen) == True:
            print('NIGHTMARE')
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False
        pygame.display.update()
    
```

This is a function for the mode screen which runs after the user has clicked on start in the menu screen. If the user left clicks on the return image in the corner of the screen, they will be returned to the main menu. If the user presses either of the other buttons on the screen then it will just print different things out for now. However, later on these if statements will be used to decide what map should be loaded in and which version of the hunter.

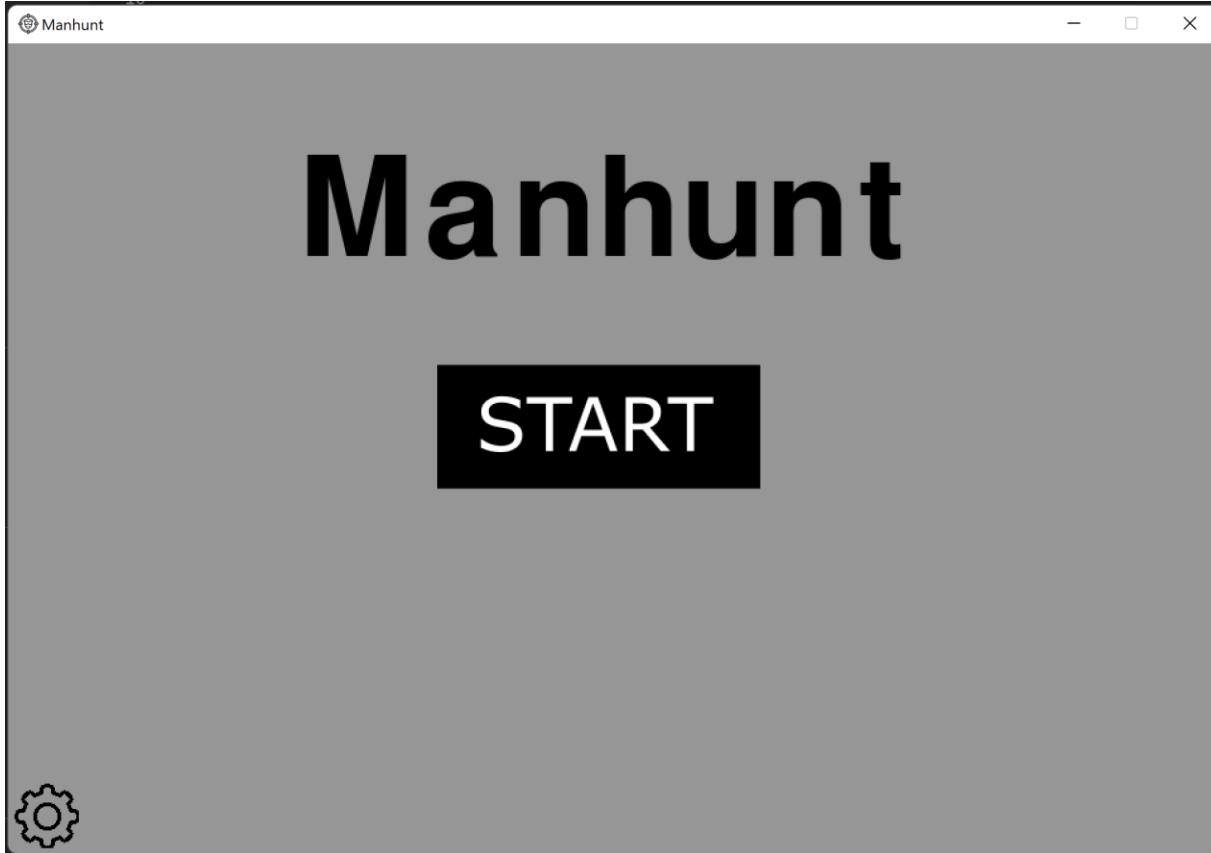
```

#Variables for mode class
mode_texts = ['Mode']
mode_textSizes = [160]
mode_textColours = [black]
mode_textFonts = [None]
mode_textCoords = [(320, 80)]
mode_images = []
mode_imagesCoords = []
mode_imageScales = []
mode.renderMTexts(mode_texts, mode_textSizes, mode_textColours, mode_textFonts, mode_textCoords)
mode.createButton('return', 'return.png', 10, 10, 0.1)
mode.createButton('normal', 'normal.png', 340, 175, 1)
mode.createButton('nightmare', 'nightmare.png', 283, 375, 1)
mode.addImages(mode_images, mode_imagesCoords, mode_imageScales)
    
```

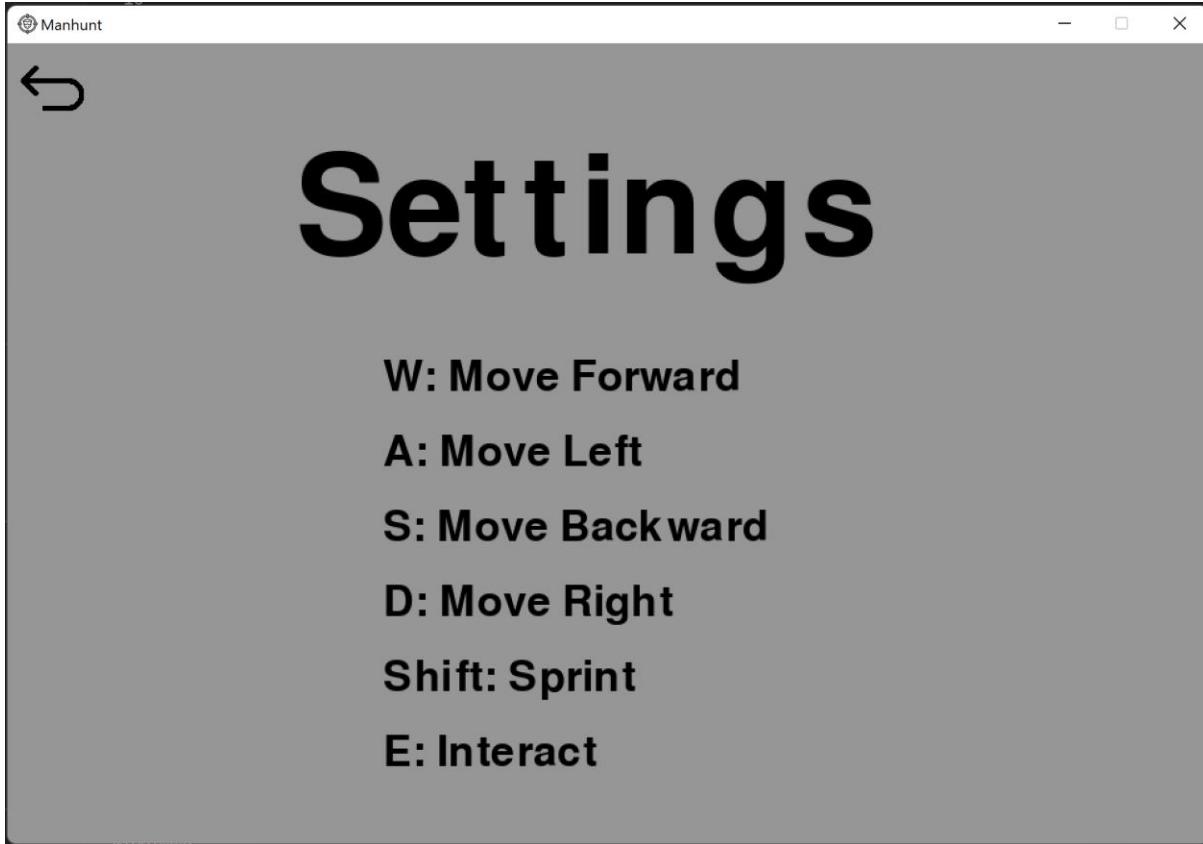
I have now created the mode screen with all the different things that need to be displayed on the screen

```
def modeScreen(mode):
    running = True
    while running:
        mode.displayScreen()
        if mode.searchButton('return').clickCheck(mode.screen) == True:
            return True
        if mode.searchButton('normal').clickCheck(mode.screen) == True:
            print('normal')
        if mode.searchButton('nightmare').clickCheck(mode.screen) == True:
            print('NIGHTMARE')
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False
        pygame.display.update()
```

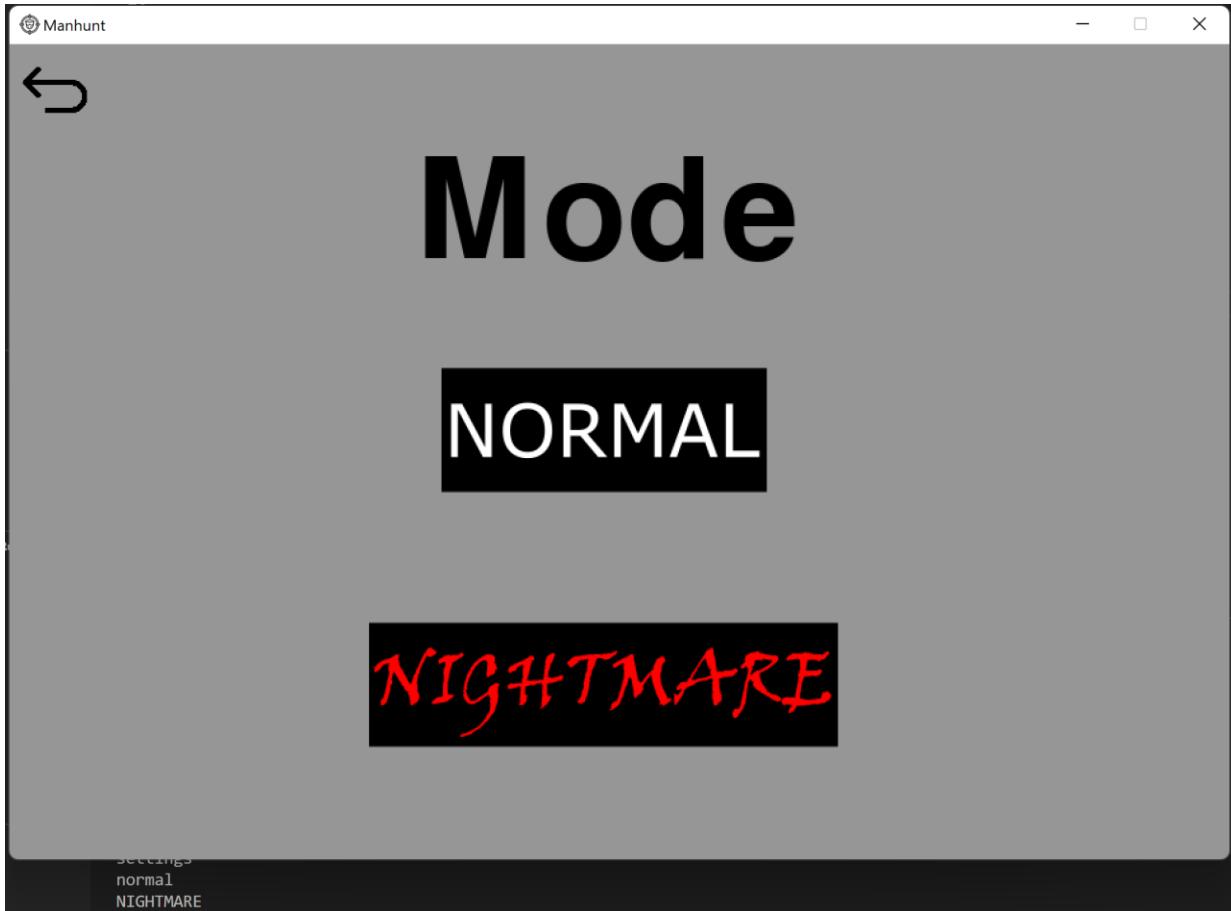
This is the procedure for displaying the mode screen which runs in a similar way to the other screens from before. However, a problem that has resulted is that when the start button is pressed on the main menu screen then the normal button is also pressed when it switches to the mode screen and prints normal straightaway. This shouldn't be happening as it would mean that the user would not be able to choose the other mode. However, this will be dealt with in other versions later on as it is not that important at the moment.



I have cleaned up the main menu screen so that only the necessary images and text show up on the screen.



For settings, I have added in the different controls so that the user can see what the controls without having to figure them out. I also added in a return button which takes the user back to the main menu screen when they want.



This is the new mode screen, the issue is shown in the text being printed outside the screen as I haven't clicked on the normal button yet but it has already printed out 'normal'. However, the printed nightmare text at the bottom was a test to see if that button works as well. I also added the return button to this screen in case the user wanted to return to the main menu screen.

Review:

In this iteration, I learnt how to create multiple game loops using subroutines in order to display multiple different screens. I also learnt how I can use my button and screen class in order to display the multiple different screens which I will need to display so that the user is able to navigate through the different screens and also to the game.

Specs completed/removed:

- P1 – now user is able to go through each of the screens including the settings screen

Specs ongoing:

- P19 – Although there is another screen where the user is able to select the difficulty, it still has issues which means that the difficulties cannot be selected

Specs started:

- P2, P7, P12, P18, P24, O6, O10, O12, Ae1, Ae5

Version 1.4:

In this version of the program I will make the GameScreen class so that the game can be displayed on the screen

```
class GameScreen(Screen):
    def __init__(self, colour):
        super().__init__(colour)
        self.colour = colour
        #self.player = player.player()
        #self.hunter = hunter.hunter()
```

This is the GameScreen class which inherits all of the attributes and methods of the parent class screen. I have commented the player and hunter attributes as I have not yet started that part and need to figure out if I need attributes for them.

```
def displayLeverCounter(self, player):
    maxLevers = player.getMaxLevers()
    activatedLevers = player.getActivatedLevers()
    renderText = self.renderText((str(activatedLevers) + '/' + str(maxLevers)), 100, (0,0,0), None)
    self.screen.blit(renderText, (self.getWidth/2, self.getHeight/2))
```

This is the display lever counter method which gets the maximum number of levers from the player and then gets the number of activated levers and sets both as variables. Then it renders the text and displays it on the screen, however, I may remove this in later iterations as I may use a different technique in so that the player knows how many levers they have activated.

```
def displayRect(self, rect, colour):
    #Draws a rect on the screen
    pygame.draw.rect(self.screen, colour, rect)
```

This is the displayRect method which is passed the parameters rect and colour. Rect is an object made using the pygame library and is a rectangle, and the colour is a tuple containing 3 integers between 1 and 255 which will decide the colour of the rect. Then a pygame method is used and is passed the self.screen attribute which is where the rectangle will be drawn/displayed and passed it the colour and the rect which needs to be displayed.

```
def displayGameScreen(self, map):
    #Displays the screen and the map
    self.displayScreen()
    for row in map:
        for object in row:
            if object != 0:
                self.displayRect(object.getRect(), self.checkObjectColour(object))
```

This is the method which displays the game screen. It only needs 1 parameter which is map. A nested for loop is used to go through each object in the map and then a validation check is done to ensure that no empty spaces are left in the map, as the empty spaces are 0s in the map list. Then the displayRect method is run and is passed the rect of that object and then another method is run which checks what colour that object should be and returns that colour which is then also passed into this method.

```
def checkObjectColour(self, object):
    if object.getVisible() == True:
        if isinstance(object, objects.Wall):
            item = object.getItem()
            if item == None:
                return colours.darkGrey
        elif isinstance(item, objects.HidingSpace):
            return colours.blue
        elif isinstance(item, objects.Lever):
            if item.getActivated() == True:
                return colours.green
            else:
                return colours.red
        elif isinstance(object, objects.Floor):
            type = object.getType()
            if type == 'carpet':
                return colours.navy
            elif type == 'concrete':
                return colours.lightGrey
            elif type == 'wood':
                return colours.lightBrown
        elif isinstance(object, objects.Door):
            if object.checkDoorActivation() == True:
                return colours.white
            else:
                return colours.darkBrown
    else:
        return colours.black
```

This is the checkObjectColour method as stated above, it is passed the object and then if statements are used to decide what colour each object should be. As the player should only be able to see what is around them, the default colour of every object is black, if their visible attribute is set to False. However, if it is set to True then the if statements use a built-in python method which returns a boolean value based on whether the object passed is of a specific class type. Both the object and the class are passed in as parameters. Depending on what object and what state or what type it is, a different colour is returned. These colours are contained in a separate file called colours.

```
black = (0, 0, 0)
red = (155, 0, 0)
white = (255, 255, 255)
lightGrey = (150, 150, 150)
green = (0, 255, 0)
blue = (0, 0, 255)
navy = (0, 0, 128)
lightBrown = (199, 150, 98)
darkBrown = (79, 60, 39)
darkGrey = (100, 100, 100)
```

These are the colours that the colours file has, all are contained as tuples with different integers for red, green and blue values.

Review:

In this iteration of the program, I learnt how I can use inheritance in order to be more efficient and save time since I was able to create a more unique class for displaying my game with the extra methods which are more particular to that screen such as the checkObjectColour method since this is not needed in any other of the screens since those objects are not displayed or used in the other screens.

Specs completed/removed:

- Ae5 – different objects are different colours and the objects which the player can interact with are a colour which the player can easily differentiate from all of the other colours.
- Ae1 – I decided to remove this as I believe that it is not a necessity to the game since the player can figure out that the hunter is chasing them by seeing the hunter following them

Name: Muqtasid Zayyan Dar

Project title: Manhunt

- P2 – I have decided to remove this as the time taken for the game to load is negligible which means that there would be no reason to implement the loading screen if the user is not going to be able to even see it

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19

Specs started:

- None

Version 2:

In this version I am going to code the classes for the different things that will be in the map which is: door class, floor class, wall class, lever class and the hiding spaces class. These different classes will create different types of objects which will then later on be fed into the Map class which will be able to create a map by using these classes to generate each type of object in the correct place. Although each class will be somewhat different, they all will need a x and y coordinate for the object and will also need a method to set each of the different attributes for the object.

Version 2.1:

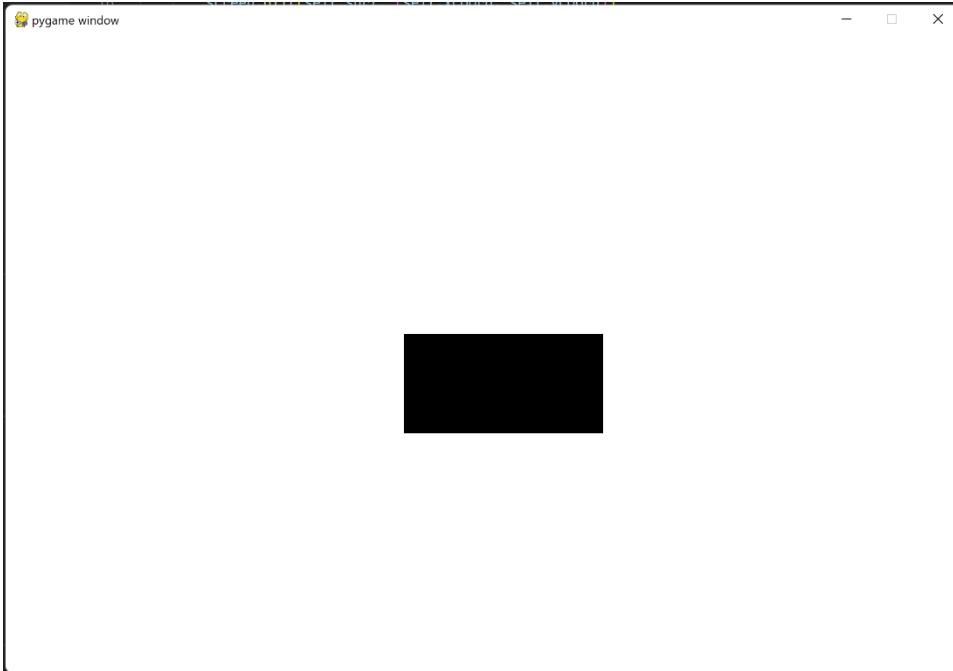
In this version I am going to code the class which all of these objects will inherit from as they have quite similar properties.

```
class Object():
    def __init__(self, x, y, height, width):
        self.xCoord = x
        self.yCoord = y
        self.height = height
        self.width = width
        self.surf = pygame.Surface([self.width, self.height])

    def getCoords(self):
        return (self.xCoord, self.yCoord)

    def display(self):
        screen.blit(self.surf, (self.xCoord, self.yCoord))
```

This is the class for an object, it may be subject to change however, this is what it looks like for now. The coordinates for the object and the height and width of the object are passed, the coordinates can be returned if needed and the surface can be displayed on the screen using the coordinates passed through.



When an object of type Object is created with parameters 400 for the x coordinate, 300 for the y coordinate, 100 for the height and 200 for the width and the display procedure is run then this is what is displayed on the screen. I had a few errors before this as I was trying to display this using the object rather than the surface, however, after moving the display function from outside the class to inside the class, it resolved the issue.

Review:

In this iteration of the program, I learnt how to use the pygame library in order to create a rect(angle) and display it on a specific part of the screen using the coords. I also learnt that the way coordinates work when displaying things, if not specified, then the coordinates passed for that rectangle is the coordinate of the top left of the rectangle. I also learnt that the x coordinates increase as you go towards the right like convention but the y coordinates decrease the further up you go.

Specs completed/removed:

- None

Spec ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19

Spec started:

- Ae2

Version 2.2:

In this version I am going to code each of the other classes, these are the door class, floor class, wall class, lever class and the hiding spaces class.

```
class Wall(Object):
    def __init__(self, x, y, height, width, hasLever = False):
        super().__init__(x, y, height, width)
        #This sets the wall as
        self.lever = self.setType()
        self.hasLever = hasLever

    def setType(self):
        #If type is none that means that there is nothing on the wall
        if self.hasLever == False:
            return None

        #If type is 1 then that means that there is a lever on the wall
        if self.hasLever == True:
            lever = Lever(self.x, self.y, self.height, self.width)
            return lever

    def checkLever(self):
        return self.hasLever

    def returnLever(self):
        return self.lever
```

This is the wall class, it inherits objects methods and attributes. It has the extra attributes `self.hasLever` and `self.lever`. The former is for use in the `setType` function which checks based on that attributes whether the wall needs a lever or not. This attribute is automatically set to false if not given a value which means that the wall doesn't have a lever. If this attribute is passed as true, then an object of type lever is created and attributed to `self.lever`. There are also the functions to check whether the wall has a lever which returns a boolean – true if the wall does have a lever and false if the wall doesn't have a lever. There is also a `returnLever` function (the more useful one) which will return whatever lever object there is that may be attributed to this wall

```
class Floor(Object):
    def __init__(self, x, y, height, width, type):
        super().__init__(x, y, height, width)
        self.type = type
        #Stores the level of sound for each floor object depending on the type of floor
        self.soundLevel = self.setSound()

    #Procedure sets the sound level depending on the type of the floor
    def setSound(self):
        if self.type == 'carpet':
            return 0.2

        if self.type == 'concrete':
            return 0.5

        if self.type == 'wood':
            return 0.9

    #Function returns the sound level of the floor
    def checkSoundLevel(self):
        return self.soundLevel
```

This is the class for floor, it also inherits from object class. It has the extra attributes type which is also a parameter and soundLevel which is determined by the type that is passed in as a parameter which is currently a string. Based on what self.type is, a value is returned for the soundLevel of this piece of floor and is attributed to self.soundLevel. The second function returns self.soundLevel to check or use the sound level for other calculations

```

class Lever(Object):
    def __init__(self, x, y, height, width):
        super().__init__(x, y, height, width)
        self.activated = False
        #This is the activation area of the Lever, it will be a factor of the height and width (above one)
        self.area = pygame.Surface([self.activationWidth, self.activationHeight])
        self.activationArea = self.area.get_rect()
        self.activationHeight = height * 1
        self.activationWidth = width * 1

    #This checks if the player is within the activation area
    def inArea(self, playerRect):
        pass

    #Function checks if lever has been activated
    def checkLeverActivation(self):
        return self.activated

    #These 2 procedures change the self.activated attribute
    def activate(self):
        self.activated = True

    def deactivate(self):
        self.activated = False

```

This is the lever class, it inherits from the object class, it doesn't have any extra parameters, however, it has a lot of extra attributes. These attributes are going to be used for the activation area around the lever which will not be visible but needs to be there so that it can be checked whether the player is colliding with that activation area and is, therefore, in range to activate the lever. I have left a placeholder function which will use methods from the physics class to determine whether the player is in the activation area of the Lever. The next function returns whether the lever is activated or not in boolean form. The last two procedures change what self.activated is.

```

class HidingSpace(Object):
    def __init__(self, x, y, height, width):
        super().__init__(x, y, height, width)
        self.area = pygame.Surface([self.activationWidth, self.activationHeight])
        self.activationHeight = height * 1
        self.activationWidth = width * 1

    def inArea(self, playerLocation):
        pass

```

This is the hiding space class which similarly to the one before, doesn't have any extra parameters, however, it still has the activation area attributes and function which will be used alongside methods from the physics class once that is made

```
class Door(Object):
    def __init__(self, x, y, height, width):
        super().__init__(x, y, height, width)
        self.activated = False

    #Function that returns whether the door has been activated
    def checkDoorActivation(self):
        return self.activated

    #Checks if the door should be activated depending on the max number of levers and how many have been activated
    def isActivated(self, activatedLevers, maxLevers):
        if activatedLevers == maxLevers:
            self.activated = True
```

This is the door class, which like all the others also inherits from the object class. It doesn't have any extra parameters, however, it has one extra attribute which is self.activated which has a boolean value and denotes whether the door has been activated or not. It also has 2 functions, the first of which just returns the boolean value of whether the door has been activated or not and the second one takes 2 extra parameters which are the number of activated levers and the max number of levers in the game right now and compares them. If they are equal then self.activated is changed to true which will later on be used to allow the player to escape the game.

```
wall1 = objects.Wall(240, 140, 200, 200)

wall2 = objects.Wall(250, 150, 200, 200, 0)

wall3 = objects.Wall(250, 150, 200, 200, 1)

door = objects.Door(100, 150, 100, 50)

carpet = objects.Floor(300, 500, 200, 200, 0)

concrete = objects.Floor(300, 400, 100, 100, 1)

wood = objects.Floor([200, 300, 50, 50, 2])
```

```
print('WALL')

print(str(wall1.wall))
print(str(wall2.wall))
print(str(wall3.wall))

print(str(wall1.getCoords()))
print(str(wall2.getCoords()))

print(str(wall3.wall.checkLeverActivation()))
wall3.wall.activate()
print(str(wall3.wall.checkLeverActivation()))
wall3.wall.deactivate()
print(str(wall3.wall.checkLeverActivation()))

print('DOOR')

print(str(door.checkDoorActivation()))
door.isActivated(3, 5)
print(str(door.checkDoorActivation()))
door.isActivated(5, 5)
print(str(door.checkDoorActivation()))

print('FLOOR')

print('CARPET')
print(str(carpet.checkSoundLevel()))
print('CONCRETE')
print(str(concrete.checkSoundLevel()))
print('WOOD')
print(str(wood.checkSoundLevel()))
```

```
WALL
None
<objects.HidingSpace object at 0x000001568030D2D0>
<objects.Lever object at 0x000001568030D270>
(240, 140)
(250, 150)
False
True
False
DOOR
False
False
True
FLOOR
CARPET
0.2
CONCRETE
0.5
WOOD
0.9
```

This is how I tested each of the classes. For the walls class, I tested whether it created an object each time after I passed in the correct parameters. Initially, it didn't create a hiding space object for the second wall due to a logical error which was because of the way I coded the check since I didn't know that false and also 0 are regarded as the same thing so it didn't create a hiding space objects and instead set it a value of none. However, getting the coordinates worked which was a part of the super class for all of these classes which is why I didn't test it for other classes. I tested whether activating and deactivating levers worked and printed out what it was and that was successful. Same thing with the door activation method depending on the amount of levers and max levers I passed it. When I tried to set the sound lever for each of the different type of floors, it initially didn't work and was printing out None for each of the floors. So instead of checking type for a string I changed it to check for different integers which would determine the sound level. However, this also didn't solve the problem. The issue was that I was trying to assign the self.soundLevel attribute within the subroutine instead of returning the value. So when I changed it, the correct numbers were printed.

Review:

In this iteration of the program I learnt how to accurately test all parts of the class so that I know that valid inputs give valid outputs and also I can see what errors can be caused by invalid inputs being given to the class which gives me insight on how I can validate the data.

Specs completed/removed:

- None

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, Ae2

Specs started:

- O5

Version 3:

In this version I am going to code the map class which will generate a map by creating objects and putting them in specific places to create a map.

Version 3.1:

In this sub-version, I will be coding and testing the first essential function of the map class, which should be creating an empty map shell with an outer border of walls. This empty shell will be created using a 2D list which will then be filled with 0s which will be considered open spaces.

```
class Map():
    #walls and floors are passed as dictionaries where the key
    #corresponds to the type of each and also includes the coordinates
    #door is passed as just coordinates
    #The coordinates in each of these are not the display coordinates but
    #the grid coordinates
    def __init__(self, walls, floors, doorCoord, mapLength):
        #doorCoord is the coordinates of the node with the door
        #not the display coordinates
        self.doorCoord = doorCoord
        #This is passed as a dictionary which only contains the coordinates of
        #Each type of wall
        self.walls = walls
        #This is also passed as a dictionary with similar things
        self.floors = floors
        self.map = []
        #As there needs to be an outer border of empty walls,
        #The real height and width of the map will need to be increased by 2
        self.mapLength = mapLength + 2

    def addHeight(self):
        for y in range(self.mapLength):
            self.map.append([])

    def addWidth(self):
        for y in range(self.mapLength):
            for x in range(self.mapLength):
                self.map[y].append(0)
```

This is the first version of the map class, it is passed a dictionary for the walls and the floors. The key for those dictionaries are the type of either wall or floor and then the value for each key is a list with the coordinates for each of the respective floor or wall. A coordinate for the door is also passed and since there is only one escape door, only one coordinate needs to be passed. Only one value for the map length is passed as the map will always be a square. However, the map length needs to be incremented

by 2 so that the border of walls can be accounted for. Then there are the two procedures for adding height to the map and adding width. The add height procedure adds self.mapLength number of empty lists to the self.map list. The addWidth procedure goes through each of the lists within the lists using the nested for loop and adds self.mapLength 0s to each list in the self.map list.

```
gameMap = maps.Map(wallsDic, floorsDic, doorCoord, 5)

gameMap.addHeight()
print(gameMap.map)
gameMap.addWidth()
print(gameMap.map)
```

This is what I used to test if it was working. The first 3 parameters are just place holders at the moment, however, the map length was set to 5. After adding the height and the width each one is printed out to see if it is carrying it out correctly.

```
[[[], [], [], [], []],  
 [[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]]
```

As shown in the picture above, it correctly first appends 7 lists to the original self.maps list and then appends 7 0s to each list after the addWidth procedure is carried out.

```
def createEmptyMap(self):  
    for y in range(self.mapLength):  
        self.map.append([])  
        for x in range(self.mapLength):  
            print(str(x))  
            if y == 0 or y == self.mapLength - 1:  
                wall = objects.Wall(x, y)  
                self.map[y].append('W')  
            elif x == 0 or x == self.mapLength - 1:  
                wall = objects.Wall(x, y)  
                self.map[y].append('W')  
            else:  
                self.map[y].append(0)
```

I merged both the add height and the add width procedures to create a different procedure that just generates an empty shell. I also changed the name of the counter variables in the for loops as I realised the first one is essentially the list index is the y coordinate and the element of each list is the x coordinate. Within the second for loop I added some if statements which carry out different things depending on which list it is or which element of the list it is. The first if statement checks if the list is the first or the last list that is being appended and if it is then it will only append wall objects to those lists – this creates a border of walls for the top and the bottom of the map. The elif statement checks whether it is the first element in the list or the last element, and if either of these are true then a wall object is appended to those places instead of a 0, this creates the side borders for the map. Finally the else statement appends 0 whenever the other 2 statements are false, this creates the empty space within the map. ‘W’ is appended right now just for testing, however, after testing this will be changed back to the wall variable in each of the if statements.

```
[ 'W', 'W', 'W', 'W', 'W', 'W', 'W' ]
[ 'W', 0, 0, 0, 0, 0, 'W' ]
[ 'W', 0, 0, 0, 0, 0, 'W' ]
[ 'W', 0, 0, 0, 0, 0, 'W' ]
[ 'W', 0, 0, 0, 0, 0, 'W' ]
[ 'W', 0, 0, 0, 0, 0, 'W' ]
[ 'W', 'W', 'W', 'W', 'W', 'W', 'W' ]
```

When testing it, I printed out each of the lists within the self.map list using a for loop. As seen above, the procedure creates a border of walls around the empty space.

Review:

In this iteration of the program I learnt how I can implement a 2D array in my program by using it for the map. Although it is not a literal 2D array, it is meant to act like one, since each array within the array can only be a specific size and the outer array also must be a specific size – depending on the length of the map which is passed to the class. Now that the empty map can be created, I need to be able to add objects into the map to be able to create the level and also some methods which will return a certain type of objects

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, Ae2, O5

Specs started:

- P26

Version 3.2:

In this sub-version I will code the different methods which will use the dictionaries to add the walls and the floors to the self.map list in the correct places.

```
def addWalls(self):
    #Goes through each of the keys in the dictionary
    for type in self.walls:
        print(type)
        #Uses the key to access the list associated with each key
        #Goes through each element in the list which is the coordinates
        #of that wall
        for coord in self.walls[type]:
            #Creates a wall object
            wall = objects.Wall(coord, type)
            #Switches one of the empty spots (0) to that wall object
            self.map[coord[1]][coord[0]] = 'AW'
            print(coord[1], coord[0])

def addFloors(self):
    #Goes through each type of floor in self.floor dictionary
    for type in self.floors:
        print(type)
        #Uses the key to access coordinates of type of floor
        for coord in self.floors[type]:
            #Creates an object of type floor, passing the coordinates and type
            floor = objects.Floor(coord, type)
            #Changes an empty spot on the map to that object
            self.map[coord[1]][coord[0]] = 'F'
            print(coord[1], coord[0])
```

These are both functions that add the walls or the floors to the map. These functions use a for loop to go through each key within each respective dictionaries and then uses those keys to access each corresponding list. Each coordinate is stored as a tuple within a list which has a key in the dictionary. The nested for loop goes through each element in each of the lists. Each element is a tuple which stores the x and then the y coordinate of each of the relevant objects. Then within the nested for loop, an object of either type floor or type wall is created and then it is passed both the coordinates (a tuple) and the type of that object which is a string and determines the type of wall or floor which each may have different

characteristics. Then the corresponding place in the self.map list is replaced with that object, however, for testing purposes I have left it as a string as it is easier to read the output and check for errors.

```
def addDoor(self):
    door = objects.Door(self.doorCoord)
    self.map[self.doorCoord[1]][self.doorCoord[0]] = 'D'
```

This is the addDoor method, it is very simple and it just creates a door object and then changes the corresponding place in the self.map list to that object. Again, for testing purposes I have left it as a string rather than the corresponding object for now.

```
wallsDic = {'empty':[(1,1)], 'hidingSpace':[(2,2)], 'lever':[(3,2)]}
floorsDic = {'wood':[(1,4),(5,4)], 'concrete':[(1,5),(2,5)], 'carpet':[(4,3)]}
doorCoord = (3 ,6)

gameMap = maps.Map(wallsDic, floorsDic, doorCoord, 5)
gameMap.createEmptyMap()
gameMap.addWalls()
gameMap.addFloors()
gameMap.addDoor()
for y in gameMap.map:
    print(y)

['W', 'W', 'W', 'W', 'W', 'W', 'W']
['W', 'AW', 0, 0, 0, 0, 'W']
['W', 0, 'AW', 'AW', 0, 0, 'W']
['W', 0, 0, 0, 'F', 0, 'W']
['W', 'F', 0, 0, 0, 'F', 'W']
['W', 'F', 'F', 0, 0, 0, 'W']
['W', 'W', 'W', 'D', 'W', 'W', 'W']
```

I used the code above to test whether my program was working or not. When I created a map of size 5 then the program creates an empty 5 by 5 grid with empty space in the middle and a border of walls around it. Then as in the dictionary, there is a wall at coordinates 1,1; 2,2, and 3,2; the floors are also correctly placed in the correct positions along with the door. The way the coordinates work in this grid is different from the conventional way. The Y coordinate starts at 0 from the top and then increases as you go down and the x coordinate starts at 0 at the left and then increases towards the right. When I first tested this I found that the coordinates of the objects were not correctly corresponding to the places I wanted them to, however, the solution to this was to switch around the way I accessed the coordinates as when I access the self.map list, each list is a y coordinate and each element in the list is an x coordinate, therefore, whenever going through the map list I would have to use the y coordinate first to access the relevant list before using the x coordinate to access the relevant element

```
#Function that returns all of the walls or a type of wall
def getWallCoords(self, category = None):
    allWalls = []
    if category == None:
        for type in self.walls:
            for wall in self.walls[type]:
                allWalls.append(wall)
        return allWalls
    else:
        return self.walls[category]

#Function that returns all of the floors or all of a single type of floor
def getFloorCoords(self, category = None):
    allFloorCoords = []
    if category == None:
        for type in self.floors:
            for floor in self.floors[type]:
                allFloorCoords.append(floor)
        return allFloorCoords
    else:
        return self.floors[category]
```

These functions return the relevant coordinates for each of the objects. If a category is given then it returns the coordinates of the objects in that category in a list. If a category is not given then it just returns the coordinates of all the walls that exist in a list

```
#Function that returns all of the wall objects
def getWalls(self, category = None):
    allWalls = []
    coords = self.getWallCoords(category)
    for coord in coords:
        wall = self.map[coord[1]][coord[0]]
        allWalls.append(wall)
    return allWalls

#Function that returns all of the floor objects
def getFloors(self, category = None):
    allFloors = []
    coords = self.getFloorCoords(category)
    for coord in coords:
        floor = self.map[coord[1]][coord[0]]
        allFloors.append(floor)
    return allFloors
```

This function returns the objects themselves in a list using the functions above to get the coordinates and then using a for loop to append each object to a list and return that list.

Review:

In this iteration of the program, I have added all of the necessary get methods for the map so that I can easily get any objects from the map, however, in some situations where I need a specific type of objects, for example, the objects which the player can interact with so I need to add some methods to return those objects as well. Furthermore, in order for the map to be able to fit the screen which I am going to be using for the game, I will need to implement a height and width variable for the map class in order to control it rather than both being the same.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, Ae2, O5, P26

Specs started:

- None

Version 3.3:

In these next versions I am going to create the complete map which will be used in the final version of the game. In order to do this, in this version, I will first update the map class with a height and width attribute so that the map can fill the whole screen. Then I will also add some more methods that I believe may need to be used later on.

```
self.mapHeight = mapHeight
self.mapWidth = mapWidth
```

These are the new attributes which I swapped for the self.mapLength attribute as I wanted my map to fit the screen exactly so I added new attributes which will be passed as parameters for the height and the width of the map.

```
#Creates an empty shell for the map with an outer border of walls and empty spaces in the middle
def createEmptyMap(self):
    #Loops for as long as the length of the map is
    for y in range(self.mapHeight):
        #for every loop a new list is appended
        self.map.append([])
        for x in range(self.mapWidth):
            #for every loop a new element is appended to the new list
            if y == 0 or y == self.mapHeight - 1:
                #This if statement checks if it is the first list or the last list
                #and if it is it fills it with wall objects
                wall = objects.Wall((x, y))
                self.map[y].append(wall)
                self.borderWalls.append((x,y))
            elif x == 0 or x == self.mapWidth - 1:
                #This checks if the element is the first or the last
                #and if it is then it appends a wall object instead
                wall = objects.Wall((x, y))
                self.map[y].append(wall)
                self.borderWalls.append((x,y))
            else:
                #This appends an empty spot in the list
                self.map[y].append(0)
```

This is the new version of the createEmptyMap method as the height and width can be different values now so I needed to swap all of the places where self.mapLength was used with either of the new attributes, depending on whether the place of reference to that attribute is for the y axis, in which case the self.mapHeight will be used, or the x axis, in which case the self.mapWidth attribute would be used.

```
#Procedure that adds the door to the map
def addDoor(self):
    for doorCoord in self.doorCoords:
        door = objects.Door(doorCoord)
        self.map[doorCoord[1]][doorCoord[0]] = door
```

This is the new version of the self.addDoor method as I wanted more than 1 door in the map to make it easier for the player to be able to escape once all of the levers have been activated.

```
#Function that returns all of the walls which the player can interact with
def getItemWalls(self):

    HWalls = self.getWalls('hidingSpace')
    LWalls = self.getWalls('lever')
    interactables = []

    for HWall in HWalls:
        interactables.append(HWall)
    for LWall in LWalls:
        interactables.append(LWall)

    return interactables
```

This is the getItemWalls method which will be used to return all of the walls which the player can interact with (and therefore, have an item). The getWalls method is used for both the walls with hiding spaces and the walls with levers and then a new variable is created which will contain all of these walls. 2 separate for loops are used for each list of walls which the player can interact with and then each of those walls are added to the new interactables list. This is then returned at the end of the method as a 1D list.

```
#Function that returns the door object based on the coordinates of the door
def getDoors(self):
    doorList = []
    for doorCoord in self.doorCoords:
        door = self.map[doorCoord[1]][doorCoord[0]]
        doorList.append(door)
    return doorList
```

This is the new getDoor method which has been renamed to getDoors so that I can add multiple doors to the map. For this I just needed to use a loop which would go through each door coordinate and then get the door object from the map list. Then I added this door to a new list called doorList. This list is then returned at the end of the method as a 1D list.

```
#This returns all of the objects as 1 complete list
def getAllObjects(self):
    walls = self.getWalls()
    floors = self.getFloors()
    doors = self.getDoors()
    all = []
    for wall in walls:
        all.append(wall)
    for floor in floors:
        all.append(floor)
    for door in doors:
        all.append(door)
    return all
```

This method creates a new list where all of the objects will be stored and then uses the get methods for all of the different objects located in the map and goes through each of these lists, adding each of these objects to the new list. Once this is done, a 1D list containing all of the objects in the map is returned.

Review:

In this iteration I have added all of the relevant methods needed to return specific objects from the class, however, the actual map itself now needs to be created for the player to be able to play on. This needs to resemble the inside of a building as per the criteria which I have set. Also there needs to be a sufficient number of levers and hiding spaces so that the game does not feel oversaturated with interactable objects.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, Ae2, O5, P26

Specs started:

- None

Version 3.4:

In this version I will be creating the map itself using coordinates for each object in the map. In order to make it easier to generate coordinates for the map, rather than doing it manually for each of the coordinates, I am creating a separate algorithm which will output all of the coordinates for a certain area depending on the coordinates that I enter into the algorithm

```

while True:
    startX = int(input('Start X Coords: '))
    endX = int(input('End X Coords: '))
    startY = int(input('Start Y Coords: '))
    endY = int(input('End Y Coords: '))

    length = endX - startX
    height = endY - startY

    xCoords = []
    yCoords = []
    coords = []

    counter = 0
    while counter <= length:
        xCoord = startX + counter
        xCoords.append(xCoord)
        counter = counter + 1

    counter = 0
    while counter <= height:
        yCoord = startY + counter
        yCoords.append(yCoord)
        counter = counter + 1

    xCounter = 0
    yCounter = 0
    for xcoord in xCoords:
        x = xcoord
        for ycoord in yCoords:
            y = ycoord
            coord = (x, y)
            coords.append(coord)

    print(len(coords))
    print(coords)

```

In order to do this, I created this algorithm which takes the lowest x and y coordinates and the highest x and y coordinates as an input which are each assigned to a variable. Then the length and height of the rectangle is calculated using the x and y coordinates. Then new lists are created for the x coordinates (xCoords), y coordinates (yCoords) and the list that will be printed after the x and y coordinates have been combined. Then using a while loop which runs as long as the counter does not equal the length or the height of the rectangle, it calculates all of the x coordinates that the rectangle covers which would be the start coordinate plus the counter.

Then each of these coordinates is appended to the new list for only the xcoordinates and then finally, the counter is incremented by 1. The same loop is done for each of the y coordinates. Then a for loop is used which goes through each of the

```

import mapObjects

empty = mapObjects.empty
lever = mapObjects.lever
hidingSpace = mapObjects.hidingSpace
wood = mapObjects.wood
concrete = mapObjects.concrete
carpet = mapObjects.carpet

emptyList = []
leverList = []
hidingSpaceList = []
woodList = []
concreteList = []
carpetList = []

for emptyCoord in empty:
    x = emptyCoord[0]
    y = emptyCoord[1]
    x -= 1
    y -= 1
    coord = (x, y)
    emptyList.append(coord)

for leverCoord in lever:
    x = leverCoord[0]
    y = leverCoord[1]
    x -= 1
    y -= 1
    coord = (x, y)
    leverList.append(coord)

for hidingSpaceCoord in hidingSpace:
    x = hidingSpaceCoord[0]
    y = hidingSpaceCoord[1]
    x -= 1
    y -= 1
    coord = (x, y)
    hidingSpaceList.append(coord)

hidingSpaceList.append(coord)

for woodCoord in wood:
    x = woodCoord[0]
    y = woodCoord[1]
    x -= 1
    y -= 1
    coord = (x, y)
    woodList.append(coord)

for concreteCoord in concrete:
    x = concreteCoord[0]
    y = concreteCoord[1]
    x -= 1
    y -= 1
    coord = (x, y)
    concreteList.append(coord)

for carpetCoord in carpet:
    x = carpetCoord[0]
    y = carpetCoord[1]
    x -= 1
    y -= 1
    coord = (x, y)
    carpetList.append(coord)

print(carpetList)

```

As every single x and y coordinate was off by 1 due to the mistake above, I had to create a different algorithm which goes through each of the list of the coordinates, takes 1 off the x and the y values, creates another tuple for it and then appends it to the new list for each of the objects.

```

empty = [(24, 11), (25, 11), (26, 11), (27, 11), (28, 11), (24, 12), (24, 13), (24, 14),
          (24, 15), (24, 16), (16, 11), (17, 11), (18, 11), (19, 11), (20, 11), (21, 11),
          (16, 12), (16, 13), (16, 14), (16, 15), (16, 16), (16, 17), (16, 18), (13, 16),
          (13, 17), (13, 18), (13, 11), (13, 12), (13, 13), (6, 11), (7, 11), (8, 11), (9, 11),
          (10, 11), (10, 13), (10, 14), (3, 15), (4, 15), (5, 15), (6, 15),
          (7, 15), (8, 15), (9, 15), (10, 15), (3, 12), (3, 13), (3, 14), (1, 11), (2, 11), (3, 11),
          (1, 8), (2, 8), (3, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (12, 8), (13, 8),
          (6, 5), (6, 6), (6, 7), (3, 4), (4, 4), (5, 4), (6, 4), (13, 3), (13, 4), (13, 5), (13, 6),
          (13, 7), (16, 8), (17, 8), (18, 8), (19, 8), (20, 8), (21, 8), (22, 8), (23, 8), (24, 8), (25, 8),
          (26, 8), (27, 8), (28, 8), (17, 5), (18, 5), (19, 5), (20, 5), (21, 5), (22, 5), (23, 5), (24, 5),
          (25, 5), (26, 5), (16, 1), (16, 2), (16, 3), (16, 4), (16, 5), (22, 1), (22, 2), (21, 18), (21, 16),
          (21, 13), (8, 4), (12, 4), (19, 2), (25, 1)
]

hidingSpace = [(29, 17), (21, 12), (10, 12), (7, 19), (11, 4), (0, 1), (25, 2)]

lever = [(29, 13), (21, 17), (6, 12), (0, 13), (7, 4), (19, 1), (29, 6)]

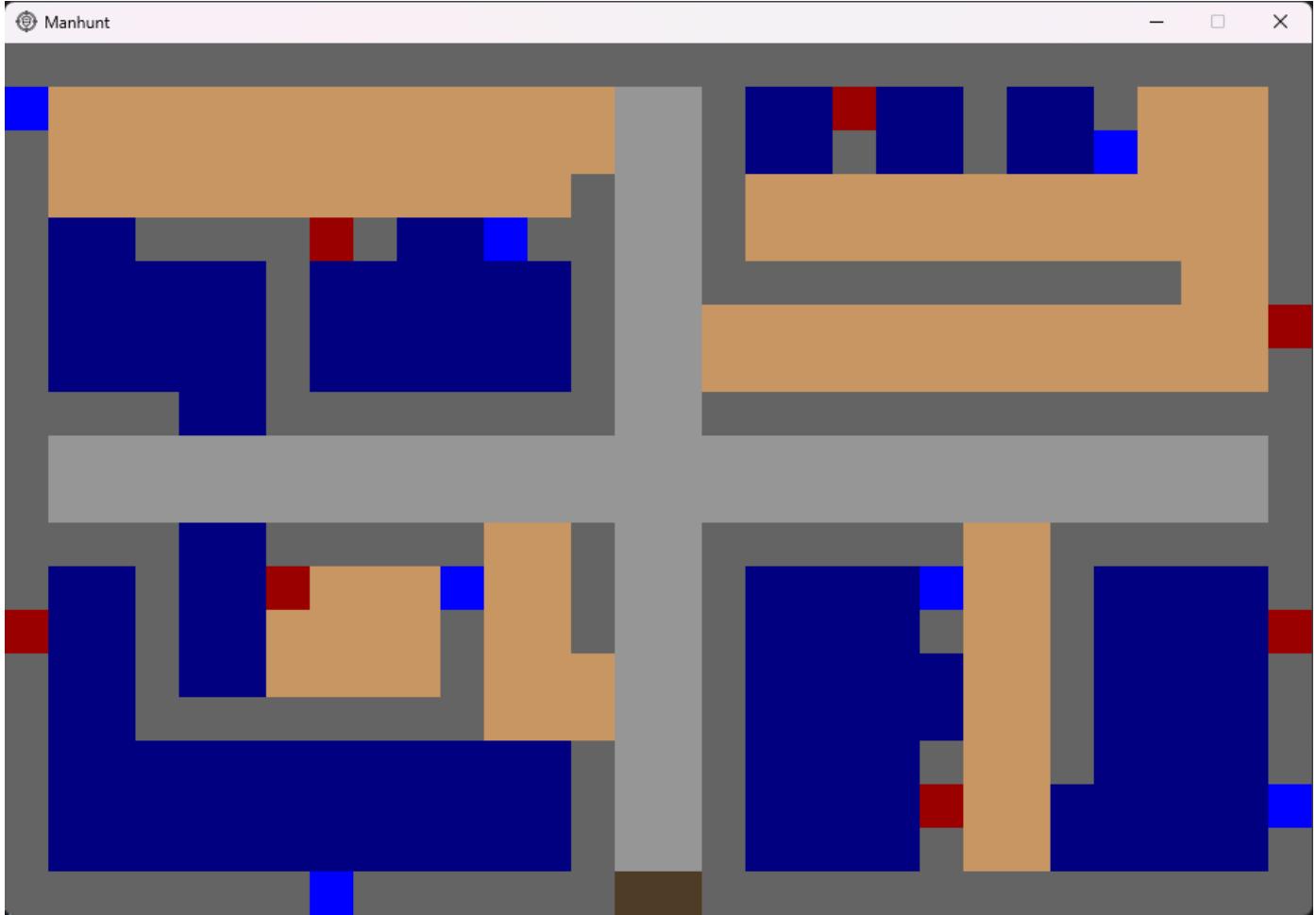
wood = [(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (4, 1), (4, 2),
          (4, 3), (5, 1), (5, 2), (5, 3), (6, 1), (6, 2), (6, 3), (7, 1), (7, 2), (7, 3), (8, 1),
          (8, 2), (8, 3), (9, 1), (9, 2), (9, 3), (10, 1), (10, 2), (10, 3), (11, 1), (11, 2), (11, 3),
          (12, 1), (12, 2), (12, 3), (17, 3), (17, 4), (18, 3), (18, 4), (19, 3), (19, 4), (20, 3), (20, 4),
          (21, 3), (21, 4), (22, 3), (22, 4), (23, 3), (23, 4), (24, 3), (24, 4), (25, 3), (25, 4), (26, 3),
          (26, 4), (27, 3), (27, 4), (28, 3), (28, 4), (16, 6), (16, 7), (17, 6), (17, 7), (18, 6), (18, 7),
          (19, 6), (19, 7), (20, 6), (20, 7), (21, 6), (21, 7), (22, 6), (22, 7), (23, 6), (23, 7), (24, 6),
          (24, 7), (25, 6), (25, 7), (26, 6), (26, 7), (27, 6), (27, 7), (28, 6), (28, 7), (22, 11), (22, 12),
          (22, 13), (22, 14), (22, 15), (22, 16), (22, 17), (22, 18), (23, 11), (23, 12), (23, 13), (23, 14),
          (23, 15), (23, 16), (23, 17), (23, 18), (11, 11), (11, 12), (11, 13), (11, 14), (11, 15), (12, 11),
          (12, 12), (12, 13), (12, 14), (12, 15), (7, 12), (7, 13), (7, 14), (8, 12), (8, 13), (8, 14), (9, 12),
          (9, 13), (9, 14), (26, 1), (26, 2), (27, 1), (27, 2), (28, 1), (28, 2), (13, 1), (13, 2), (27, 4), (28, 4),
          (13, 14), (13, 15), (6, 13), (6, 14), (27, 5), (28, 5)
]

concrete = [(1, 9), (1, 10), (2, 9), (2, 10), (3, 9), (3, 10), (4, 9), (4, 10), (5, 9), (5, 10), (6, 9),
          (6, 10), (7, 9), (7, 10), (8, 9), (8, 10), (9, 9), (9, 10), (10, 9), (10, 10), (11, 9), (11, 10),
          (12, 9), (12, 10), (13, 9), (13, 10), (14, 9), (14, 10), (15, 9), (15, 10), (16, 9), (16, 10), (17, 9),
          (17, 10), (18, 9), (18, 10), (19, 9), (19, 10), (20, 9), (20, 10), (21, 9), (21, 10), (22, 9), (22, 10),
          (23, 9), (23, 10), (24, 9), (24, 10), (25, 9), (25, 10), (26, 9), (26, 10), (27, 9), (27, 10), (28, 9),
          (28, 10), (14, 11), (14, 12), (14, 13), (14, 14), (14, 15), (14, 16), (14, 17), (14, 18), (15, 11), (15, 12),
          (15, 13), (15, 14), (15, 15), (15, 16), (15, 17), (15, 18), (14, 1), (14, 2), (14, 3), (14, 4), (14, 5), (14, 6),
          (14, 7), (14, 8), (15, 1), (15, 2), (15, 3), (15, 4), (15, 5), (15, 6), (15, 7), (15, 8)
]

carpet = [(1, 5), (1, 6), (1, 7), (2, 5), (2, 6), (2, 7), (3, 5), (3, 6), (3, 7), (4, 5), (4, 6), (4, 7), (5, 5),
          (5, 6), (5, 7), (7, 5), (7, 6), (7, 7), (8, 5), (8, 6), (8, 7), (9, 5), (9, 6), (9, 7), (10, 5), (10, 6),
          (10, 7), (11, 5), (11, 6), (11, 7), (12, 5), (12, 6), (12, 7), (17, 1), (17, 2), (18, 1), (18, 2), (20, 1),
          (20, 2), (21, 1), (21, 2), (23, 1), (23, 2), (24, 1), (24, 2), (25, 12), (25, 13), (25, 14), (25, 15), (25, 16),
          (25, 17), (25, 18), (26, 12), (26, 13), (26, 14), (26, 15), (26, 16), (26, 17), (26, 18), (27, 12), (27, 13), (27, 14),
          (27, 15), (27, 16), (27, 17), (27, 18), (28, 12), (28, 13), (28, 14), (28, 15), (28, 16), (28, 17), (28, 18), (17, 12),
          (17, 13), (17, 14), (17, 15), (17, 16), (17, 17), (17, 18), (18, 12), (18, 13), (18, 14), (18, 15), (18, 16), (18, 17),
          (18, 18), (19, 12), (19, 13), (19, 14), (19, 15), (19, 16), (19, 17), (19, 18), (20, 12), (20, 13), (20, 14), (20, 15),
          (20, 16), (20, 17), (20, 18), (4, 16), (4, 17), (4, 18), (5, 16), (5, 17), (5, 18), (6, 16), (6, 17), (6, 18), (7, 16),
          (7, 17), (7, 18), (8, 16), (8, 17), (8, 18), (9, 16), (9, 17), (9, 18), (10, 16), (10, 17), (10, 18), (11, 16), (11, 17),
          (11, 18), (12, 16), (12, 17), (12, 18), (4, 11), (4, 12), (4, 13), (4, 14), (5, 11), (5, 12), (5, 13), (5, 14), (1, 12),
          (1, 13), (1, 14), (1, 15), (1, 16), (1, 17), (1, 18), (2, 12), (2, 13), (2, 14), (2, 15), (2, 16), (2, 17), (2, 18), (4, 8),
          (5, 8), (1, 4), (2, 4), (21, 14), (21, 15), (24, 17), (24, 18), (9, 4), (10, 4), (3, 16), (3, 17), (3, 18)
]

doors = [(15, 19), (14, 19)]

```



This is what happens when the program is run with all of these coordinates entered as each of the objects. This is the output I expected as no objects are missing from the map and every object is in the right place. Also, all of the objects are displayed with the correct colours: deactivated levers are red; hiding spaces are blue; walls are dark grey; concrete floor is light grey; doors are brown; wood floor is light brown; and carpet floor is dark blue. However, I may change the colour of the carpet since it is difficult to see the player when the player image is displayed on the screen since the player and the carpet are both dark colours. Therefore, the success criteria has been completely met for this test.

Review:

In this iteration of the program, I have learned that I should always double check code for logical errors and make sure that I stay consistent with things throughout the program as in this case, I started from 1 for the coordinates in the program but started from 0 for the other program.

Specs completed/removed:

- P26, Ae2 – This has been completed as the level now has a design which resembles the inside of a building – although there may be some dead ends within the level, there are hiding spaces next to all of those places so the player can still escape from the hunter even if they get chased towards that area

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5

Specs started:

- None

Version 4:

In this version I am going to code the physics class and the projectiles class which will be inheriting the physics class.

Version 4.1:

In this sub version I am going to code something so that I can test how collisions can be implemented into my game and then use different parts of this to create something I can use for my physics and projectile classes

```
#Testing rectangles collisions
movingRect = pygame.Rect(25, 50, 80, 80)
otherRect =pygame.Rect(150, 300, 400, 100)

def setSpeed(rect, screen):
    global otherSpeed, xSpeed, ySpeed
    rect.x += xSpeed
    rect.y += ySpeed
```

First I created some rect(angle)s which are different sizes and start on different places on the screen. Then I defined a new procedure which is going to be what handles all of the physics. It takes in the parameters of rect (which is one of the rectangles) and the screen this needs to be displayed on. The variable otherSpeed is the vertical speed of the otherRect and the xSpeed and ySpeed are for the movingRect. I made these variables global as the rects are declared globally so for the speed variables and the collisions to take affect, the variables need to be global otherwise, the rects just go through each other.

```
#Collision with screen borders
if rect.right >= screen.getWidth() or rect.left <= 0:
    xSpeed *= -1
    print(xSpeed)
if rect.top <= 0 or rect.bottom >= screen.getHeight():
    ySpeed *= -1
    print(xSpeed)

#Moving other rect
otherRect.y += otherSpeed
if otherRect.top <= 0 or otherRect.bottom >= screen.getHeight():
    otherSpeed *= -1
```

This is the next part within the procedure, the top 2 'if' statements make sure that when the movingRect collides with the sides of the window that it bounces back by inverting the axial speed when colliding with the relevant border. It checks if the right side of the rect is greater than the width of the screen which implies that the rect has collided on the right of the screen, therefore, the xSpeed needs to be inverted so that it looks like the rect has bounced off the side of the window and is going in the other direction. The second condition in the if statement checks if the left of the rect has collided on the left side of the window as the x coordinate of the window starts from 0 we can check if the left side of the rect has collided with the point where the value of the x coordinate of the screen is 0 and then also inverse the xSpeed. The next if statement does the same but with the ySpeed. The part after that moves the otherRect and then does a similar thing by checking if the otherRect collides with the top or the bottom of the screen and then inverts the speed of the otherRect.

```
#Collision with other rects
collisionTolerance = 10
#Checking if the moving rectangle has collided with another rectangle
if movingRect.colliderect(otherRect):
    #Each one of these checks in what way the rectangles have collided with each other
    #As the 'movingRect' must go in the opposite direction when it hits a different object, to know which
    #direction to move it in, we need to check how one rectangle has collided relative to the other
    #to do this we compare the absolute values of the difference of the opposite sides of each of the rectangles
    #to the collisionTolerance (which is how close one rect must be to the other in pixels to be considered a collision)
    #and we also need to check in which direction the object is moving by comparing the speed of the object in a specific direction
    #as one of the objects may have collided with another object and they are moving in the same direction which could cause the object
    #to move in the opposite direction even though it should still continue in the same direction
    if abs(otherRect.top - movingRect.bottom) < collisionTolerance and ySpeed > 0:
        ySpeed *= -1
        print(ySpeed)
    if abs(otherRect.bottom - movingRect.top) < collisionTolerance and ySpeed < 0:
        ySpeed *= -1
        print(ySpeed)
    if abs(otherRect.left - movingRect.right) < collisionTolerance and xSpeed > 0:
        xSpeed *= -1
        print(xSpeed)
    if abs(otherRect.right - movingRect.left) < collisionTolerance and xSpeed < 0:
        xSpeed *= -1
        print(xSpeed)

#This redraws the 'movingRect' - 'otherRect' is drawn in main game loop
pygame.draw.rect(screen.getScreen(), (0,0,0), rect)
```

This part checks if the movingRect has collided with the other rect and then also checks in what way both of the rects have collided, based on that it inverts one of the speeds of the object so that it bounces away, most of the details of the logic is explained in the comments in the if statement.

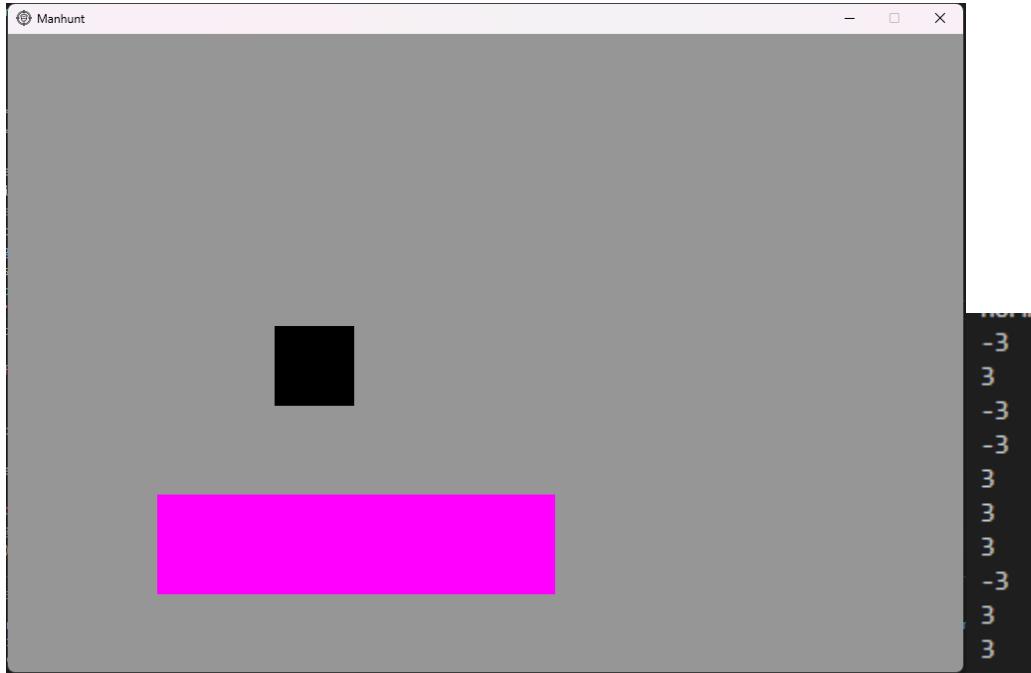
```
xSpeed = 3
ySpeed = 3
otherSpeed = 2

def gameScreen(clock):
    running = True
    while running:
        clock.tick(60)
        game.displayScreen()
        setSpeed(movingRect, game)
        pygame.draw.rect(game.getScreen(), (255, 0, 255), otherRect)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False
        pygame.display.update()
```

These are the variables that are for the speed of the movingRect and the otherRect and then the game loop which I have used to test the physics and see if it works. The game loop runs the same as for the

pprecious screen, however, this time within the loop we invoke the set speed procedure which controls the collisions and the speed of each of the two rectangles.



This is testing the physics, the rectangles on the screen are moving, the black one has a horizontal and vertical speed of 3 and the pink rectangle has no horizontal speed and a vertical speed of 2. I added some print statements whenever a collision happens between the rectangles and the direction of the speed of the black rectangle changes. On the right are the prints while the program is running which shows that the rectangles speed are changing. As the direction change is only coded for the black rectangle, only the black rectangle travels in the opposite direction after a collision with either the borders of the screen or the pink rectangle. The pink rectangle only has collisions between the borders of the window coded in.

Review:

In this version I learned how to handle projectiles and collisions using pygame and calculations. The techniques which I learnt here will be used later on to influence the logic in other parts of my program such as the projectiles, player and the hunter as these parts of the program all will deal with moving objects and collisions. Now that I have learnt this I can start creating the projectiles which will be used for sight and sound.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5

Specs started:

- O1, O2, O3, O4 – These are linked with the players movement and the code I have created helps me realise how I can implement these for the player as well

Version 4.2:

In this version I am going to code the parent projectile class which will be inherited by 2 more projectile classes which will be used for more specific things

```
#Parent class for sight projectiles and sound projectiles
class Projectile():
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__()
        #Control the speed of the projectile
        self.xSpeed = xSpeed
        self.ySpeed = ySpeed
        #Origin coordinates of the projectile
        self.coords = coords
        #This is the actual rect for the projectile, created using pygame library
        self.rect = pygame.Rect(coords[0], coords[1], length, length)
        #This is a boolean value which is whether the projectile has collided with a wall or not
        self.collided = False
        #This is a boolean value which is whether the projectile has been launched or not
        self.launched = False
```

This is the projectile class, it takes parameters, coords – a tuple where the projectile will originate from, xSpeed – speed in the x direction, ySpeed – speed in the y direction and then length – this determines the size of the projectile (length of the height and width of the projectile). The first 3 parameters are set as attributes, and then length is used for the self.rect attribute which contains the rect created using the pygame library. This rect is passed the coordinates (coords) and then the length of the height and width which I kept the same as I wanted the projectile to be a square. The self.collided attribute and the self.launched attribute are set as false. The self.collided attribute will be used to check if the projectile has collided with a wall and will be set to true when that happens and then the self.launched attribute is to check whether the projectile has been launched for the first time and is kept as True after the first launch.

```
#This is a get function which returns the value for the launched attribute
def getLaunched(self):
    return self.launched

#This is the get function returning the value for the collided attribute
def getCollided(self):
    return self.collided

#This is a function that checks if an object passed has collided with itself
def collideCheck(self, object):
    return self.rect.colliderect(object)
```

These are the first few methods of the projectile class. The first two methods are get methods for the launched and collided attributes which return their respective attributes. The next method takes a parameter called object, which will be an object such as a wall, floor or a door. Then a pygame method will be used on the rect attribute of the projectile and passed the object which will return a boolean value based on whether the projectiles rect and the object passed into the pygame method have collided or not, then this result is also returned via the collideCheck method.

```
#This is a procedure that sets the value of the collided attribute as a
#value passed to the procedure
def setCollided(self, value):
    if value == True or value == False:
        self.collided = value

#This is the procedure that moves the projectile, screen is for testing purposes
def moveProjectile(self, screen):
    self.rect.x += self.xSpeed
    self.rect.y += self.ySpeed

    pygame.draw.rect(screen.getScreen(), colours.red, self.rect)
```

These are the last 2 methods in the projectile class. The first method is a set method for the collided attribute and based on the value that is passed into the method, if it is a boolean value, then the collided attribute is set as that value. The second method is passed the parameter screen, which is the screen on which the projectile will be displayed (for testing) . However, the main use of this method is to move the projectile across the screen by adding on the xSpeed or ySpeed attributes to the x or y coordinates of the projectile rect. Based on the value of the xSpeed or ySpeed attributes, every time the method is invoked, the projectile is moved across the screen by that many pixels.

Review:

In this iteration of the program I have now coded the projectile class which will be inherited by the sound and sight projectile classes which will be coded later one, by coding this, I have learnt how to think ahead so that I can figure out what both projectile classes may have in common which I would need to add to this parent projectile class.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, O1, O2, O3, O4

Specs started:

- P8, P9, P20, P21, O11

Version 4.3:

In this version I am going to create the projectile class which will be used for the player's field of view while they are playing.

```
#This is the class for sight projectiles, which inherits projectile
class SightProjectile(Projectile):
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__(coords, xSpeed, ySpeed, length)
        #This attribute is a list which will contain all of the objects that the projectile has
        #Collided with
        self.collidedObjects = []
```

The SightProjectile class inherits all of the attributes and takes the same parameters as the parent projectile class. However, it has the extra attribute of collidedObjects, which is in the form of an empty list. This will be used to keep a track of what objects the projectile has collided into.

```
#This returns a list of all of the objects which the projectile has collided with
def getCollidedObjects(self):
    return self.collidedObjects

#This is a method that searches through all of the objects in order to check
#If the projectile has collided with it
def searchObjects(self, searchObject):
    found = False
    for object in self.collidedObjects:
        if object == searchObject:
            found = True
    return found
```

These are the first 2 methods of the class. The first method just returns the collidedObjects attribute which is a list containing all of the objects which the projectile has collided with. Initially when I was deciding how to implement this part of the program, I created a searchObjects method which takes an object as a parameter and then uses a linear search to search for the object and then returns a boolean value based on whether the object has been found or not. However, in later iterations, I did not need this caused my program to be very slow when I used it in testing. Therefore, it will not be used in later iterations of the program.

```
#This method checks if the projectile has collided with any of the objects in the map
#and based on that it sets the visibility as True and if it is a wall then it sets
#The collided attribute as true
def objectCheck(self, allObjects):
    #Goes through all of the objects in the map
    for object in allObjects:
        rect = object.getRect()
        #Method invoked with that object above passed to check if the projectile has
        #collided with it
        if self.collideCheck(rect) == True:
            #print(self.collideCheck(rect))
            #If it has then it is added to the collided objects list
            self.collidedObjects.append(object)
            #The visibility of that object is set to True so it can be seen
            object.setVisible(True)
            #Checks if the object is a door or a wall and if it is then it has collided so collided is True
            if isinstance(object, objects.Wall) or isinstance(object, objects.Door):
                self.collided = True
            #elif self.searchObjects(object) == False:
            #object.setVisible(False)
```

This is the objectCheck method which takes the parameter allObjects – this parameter is a list of all of the objects that are in the map (walls, floors and doors). This method goes through each object in this list and then sets the variable rect as the rect of that specific object. Then a the collideCheck method is used in an if statement and is passed this rect which will return a boolean value based on whether or not the projectile has collided with that specific rect. If this has happened then this object is added to the collided objects list and its visibility is set to true so that the player is able to see the object. Then there is another if statement which checks if the object is a wall or a door and then if it is, the collided attribute of the projectile is set to true so that it doesn't continue checking any longer.

```
#This method launches the projectile and then runs the other methods in order to check
#what objects the projectile has collided with and returns the list of objects the projectile
#Has collided with
def launchSightProjectile(self, screen, map, coords):
    #Sets the collided objects as an empty list each time the projectile is launched
    self.collidedObjects = []
    #This is a list of all of the objects in the map
    allObjects = map.getAllObjects()
    #The center of the projectile is set as the coordinates passed
    self.rect.center = coords
    self.launched = True
    #Runs while the projectile has not collided with a wall or door
    while self.collided == False:
        self.moveProjectile(screen)
        self.objectCheck(allObjects)
    #Once it has collided with a wall or door then self.collided will be true
    #So list of all collided objects are returned
    return self.collidedObjects
```

This is the final method which takes the parameters, screen – this is the screen that the projectile will be displayed on and is passed into another method in this method(for testing purposes), map – which will be used to get all of the objects in the map as a list, and coords – this will be used so that the projectile can be launched from this coordinate. Each time this method is invoked then the collidedObjects attribute is reset to an empty list otherwise, each time the projectile is launched, all of the objects from

the previous launch would be kept visible as it would think that the projectile has collided with that object. Then the center of the projectile is set as the coordinate passed as a parameter which is a tuple containing the x and y values which are both integers. Then the launched attribute is set as True, once this is set as True once then it stays as True, this is so that I can check later on if the projectile has been launched for the first time. Then there is a loop which runs while the collided attribute is false, this means that the loop will run as long as the projectile does not collide with a wall or a door. Within this loop, both the moveProjectile and objectCheck methods are invoked. At the end of the method, the list of all of the objects the projectile has collided with (self.collidedObjects) is returned and will be used.

Review:

In this part of the program I have learnt how I can use the map class and the pygame collision techniques which I learnt about before in order to detect collisions between certain objects. Now that the sight projectile class has been completed, I will now need to complete the sound projectile class as that is needed for the hunter.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, O1, O2, O3, O4, P8, P9, P20, P21, O11

Specs started:

- None

Version 4.4:

In this version I am going to code the Sound Projectile class which will be used in order to check the number of walls in between the player and the hunter for the hunter's sound function.

```
#This is the sound projectile class which will be used by the hunter to determine the number of walls between the player and the hunter
class SoundProjectile(Projectile):
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__(coords, xSpeed, ySpeed, length)
        #This attribute will be used to record the number of wall which the projectile has collided into
        self.wallNum = 0
```

This is the sound projectile class which inherits the projectile class, it only has one extra attribute which is wallNum and counts the number of walls which the projectile has collided with.

```
#This method checks if the projectile has collided into the player by taking the player's hitbox as a parameter
def playerCollision(self, player):
    if self.collideCheck(player.getHitbox()):
        self.setCollided(True)
    self.setCollided(False)
```

This is the player collision method and it checks if the sound projectile has collided with the player's hitbox, if it does then the collided value is set to True, otherwise, the collided value is kept as false

```
#This method launches the sound projectile and then returns the number of walls the projectile has collided into
def launchSoundProjectile(self, screen, coords, map, player):
    self.rect.center = coords
    walls = map.getWalls()
    while self.collided == False:
        self.moveProjectile(screen)
        self.wallCheck(walls)
        self.playerCollision(player)
        print(self.collided)
    print(self.collided)
    collidedNum = self.wallNum
    self.wallNum = 0
    return collidedNum
```

This is the launchSoundProjectile method which takes the parameters: screen – a surface object, coords – a tuple, map – a map object and player – a player object. Then it sets the coordinates of the center of the projectile as the coordinates passed in, next a method is used on the map object in order to get all of the walls within the map as a list. Then a while loop is used which runs while the collided attribute is false which means the projectile has not collided with the player. Within this loop, the first line moves the projectile across the screen, then the next method checks whether the projectile has collided with a wall, then the last method checks if the projectile has collided with the player. The 2 print statements after that are for testing. Then a temporary variable is used to store the wallNum attribute which is then set to 0 and the temporary variable is returned.

```
def wallCheck(self, walls):
    for wall in walls:
        if self.collideCheck(wall) == True:
            self.wallNum += 1
```

This is the wall check method which just checks if the projectile has collided with any of the walls on the map and if it has then it increments the wall number by 1

Review:

In this iteration of the program I have learnt how to apply the techniques which I learnt from the sight projectiles class in a different way in order to create the sound projectiles class as it functions in a similar way to the sight projectile class but has a completely different function. Now that the main parts of the game the player and hunter needs in order to be able to function exist (map and projectiles) I can now create the player and hunter.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, O1, O2, O3, O4, P8, P9, P20, P21, O11

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Specs started:

- None

Version 5:

In this version I am going to code the player class which will be inheriting the sprite class from the pygame library. The player will be able to move forwards, backwards, left and right using the W, A, S and D keys on the keyboard. The player will also be able to interact with different objects around the map by pressing the E key while near that object. If the player presses the Shift button while moving then their speed should be increased and they should be moving quicker on the screen.

Version 5.1:

In this sub version I am going to code the basic necessities of the players class which will include some methods for movement, displaying the player and creating getters and setters for the coordinates of the player

```
class Player(pygame.sprite.Sprite):
    def __init__(self, x, y, img, scale, speed):
        super().__init__
        #Coordinate attributes
        self.x = x
        self.y = y
        self.screenCoords = (x, y)
        self.mapCoords = (x/32, y/32)
        #Image attributes
        self.img = pygame.image.load(img).convert_alpha()
        width = self.img.get_width()
        height = self.img.get_height()
        self.transformedImg = pygame.transform.scale(self.img, (int(width * scale), int(height * scale)))
        #Speed in any direction
        self.speed = speed
        #Hitbox of the player by creating a rectangle around the player img
        self.hitbox = self.transformedImg.get_rect()
        self.center = self.hitbox.center
        self.hiding = False
```

This is the start of the player class, it takes the parameters x, y img, scale and speed, it also inherits from the sprite class which is a part of the pygame library. The x and y coordinates are for where the player will spawn when the game is first run, the img parameter is for the name of the image so that it can be displayed on the screen later on, the scale is also to do with the image so that its size can be changed without having to use a different version of the image. The speed parameter is so that the speed of the player can easily be changed by changing the parameter that is passed to the player class. The first few attributes are for the coordinates on the player, whether it is relevant to the speed or the map itself, the screenCoords attribute is for the player on the screen itself and the mapCoords are for the coordinates of the player on the map, the x and y coordinate are divided by 32 as this is the factor between the map and the screen. the next section is all to do with the player image on the screen, self.img is the loaded image of the player which also has the .convert_alpha() method run on it from the pygame library so that it can be used later on if I want to make it seem as the player is not on the screen anymore which could be when the player is hiding. The width and height variables are the height and width of the player image and also for the attribute after that which transforms the image so that it is easier to control the size of the image without having to find the same image at a different size. These are then used along with the scale passed as a parameter through the constructor which then transforms the image based on the scale (less than 1 makes the image smaller, greater than 1 makes the image bigger). The

`self.speed` attribute is the parameter that is passed for speed which is used to move the character. The next attribute is `self.hitbox` which is meant to be a rectangle the size of the player image and `self.center` is meant to be the center of the hitbox which I was thinking could be used for the coordinates as previous coordinates are from the top left of the image which could cause problems. The final attribute so far is the `self.hiding` attribute which is a boolean value and may be used later on to check if the player is hiding or not and carry out actions based on that fact.

```
#Displays the player on whatever screen is passed to the method
def displayPlayer(self, screen):
    screen.blit(self.transformedImg, self.screenCoords)
```

This is the `displayPlayer` procedure which simply just displays the player image on the screen using the transformed player image and the screen coordinates when invoked

```
#Checks if the W key has been pressed and if it has then it deducts 1 from the player coordinates and update
#the coordinates of the player
def moveForward(self, pressed):
    if pressed[pygame.K_w]:
        self.y -= self.speed
        self.setCoords()
        #print('forward')

#Checks if the S key has been pressed and if it has then it increments the players coordinates by 1 and updates
#the coordinates of the player
def moveBackward(self, pressed):
    if pressed[pygame.K_s]:
        self.y += self.speed
        self.setCoords()
        #print('backward')

#Checks if the D key has been pressed and if it has then it increments the players coordinates by 1 and updates
#the coordinates of the player
def moveRight(self, pressed):
    if pressed[pygame.K_d]:
        self.x += self.speed
        self.setCoords()
        #print('right')

#Checks if the A key has been pressed and if it has then it deducts 1 from the players coordinates and updates
#the coordintes of the player
def moveLeft(self, pressed):
    if pressed[pygame.K_a]:
        self.x -= self.speed
        self.setCoords()
        #print('left')
```

These 4 method control the player's movement, they all take `pressed` as a parameter from a procedure later on and function in a very similar way. The general way each one works is that it checks if a specific button on the keyboard has been pressed and then based on that it carries out an action on the `x` or `y` coordinate of the player, which either increases it or decreases it and the calls the `setCoords()` function which just updates the coordinates.

```

#This changes the coordinates of the player whenever it is called, as the map coordinates of the player are directly linked to the
#screen coordinates of the player, by changing the screen coordinates (which moves the player image and rect) it also changes the
#players location on the other map
def setCoords(self):
    self.screenCoords = (self.x, self.y)
    #print(str(self.screenCoords))

def setMapCoords(self):
    x = round(int(self.screenCoords[0])/32)
    y = round(int(self.screenCoords[1])/32)
    self.mapCoords = (x, y)

#This method checks if the W, A, S, D, E, or LEFT SHIFT buttons are pressed and then carries out various actions depending
#on the button that is pressed
def checkKeys(self, map):
    #Uses a pygame method to check if any key has been pressed and will be true if it is pressed and false if no buttons
    #on the keyboard are being pressed
    pressed = pygame.key.get_pressed()

    #Each of these methods control an aspect of the inputs from the user which in turn controls the player
    self.moveForward(pressed)
    self.moveBackward(pressed)
    self.moveRight(pressed)
    self.moveLeft(pressed)
    self.sprint(pressed)
    self.interact(pressed, map)

#Returns the map coordinates of the player
def getMapCoords(self):
    self.setMapCoords()
    return self.mapCoords

```

The first method is the `setCoords()` method which I talked about previously which just uses the `self.x` and `self.y` coords to just update them. The next function is for the map coordinates which is so that I can do things such as get the floor that the player is currently on to use in the calculation for the sound when coding the hunter, this takes each coordinate from the `self.screenCoords` tuple and then divides it by 32 to then get a value for the map coordinates. However, after writing this I believe that I may get rid of the `screenCoords` and `mapCoords` attributes and just keep the `x` and `y` attributes which can then be used to calculate map coords which will be more efficient. The next function is the `checkKeys` function which takes the map as a parameter and then it sets a new variable `pressed` as the boolean result of the `.get_pressed()` function from the `pygame` library which checks whether any keys on the keyboard has been pressed and returns true if it has been and false if no buttons have been pressed. This is then passed on to each of the functions which control player actions and based on the button, a specific action is carried out for the player. This makes it quicker later on when I want to invoke something which can check the users inputs since it invokes all of the other methods. The final method currently is the `getMapCoords` method which simply returns the map coordinates after updating them.

Review:

In this iteration of the program, I learned another way in which I could implement what I learned from creating the little physics program in version 4.1. I also learned how I could use the `pygame` library in order to get what keys the user is pressing and then use these inputs to result in corresponding actions for the player. Although the player is now able to move on the screen, I now need to test this in order to check if this works which will be done in the next section once the sprinting and interacting aspect of the player is done.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, O1, O2, O3, O4, P8, P9, P20, P21, O11

Specs started:

- I1, I2, I3, I4, I6
- Ae3, Ae4 – I am slightly changing the criterion for these spec points as I believe these sprites are still too overcomplicated so I am going to make the player appear as a circle on the screen with the hunter a different colour

Version 5.2:

```
class Player(pygame.sprite.Sprite):
    def __init__(self, x, y, img, scale, speed, sprintMultiplier):
        super().__init__()
        #Coordinate attributes
        self.x = x
        self.y = y
        #Image attributes
        self.img = pygame.image.load(img).convert_alpha()
        width = self.img.get_width()
        height = self.img.get_height()
        self.transformedImg = pygame.transform.scale(self.img, (int(width * scale), int(height * scale)))
        self.width = self.transformedImg.get_width()
        self.height = self.transformedImg.get_height()
        #Hitbox of the player by creating a rectangle around the player img
        self.hitbox = self.transformedImg.get_rect()
        #Activation area for the player to use to detect levers and hiding places around the player when the player interacts
        self.activationArea = pygame.Rect(self.x - 32, self.y - 32, self.width * 3, self.height * 3)
        #Speed in any direction
        self.speed = speed
        #This is the sprinting multiplier which increases the speed of the player by that much
        self.sprintMultiplier = sprintMultiplier
        self.hiding = False
```

This is the player class now, I have got rid of the redundant coordinates and added some new attributes to be used for the activation area and the sprinting function of the player. I have added self.width and self.height for the transformed image so that it can be used to make the activation area a specific factor of the area of the image. The self.hitbox attribute is still the same and I have just moved it around so that it is around the other similar attributes. Another new attribute is the self.activationArea attribute which creates a new rectangle around the player using a function from the pygame library and then creates it with the top left of the image being -32 pixels in both axial directions – this was so that the image would be centred in the activation area, however, a different function corrects this straightforwardly, so the coordinates do not matter that much in this situation. For now, the activation area is 9 times bigger than the image as the width and height of the original image are tripled for the activation area, this is so that the player does not have to come too close to the walls in order to use their functions. The final new attribute is the sprintMultiplier which is passed as a parameter to the class and is used to increase the movement speed of the player by a factor when the shift button is pressed on the keyboard.

```
#Displays the player on whatever screen is passed to the method
def displayPlayer(self, screen):
    pygame.draw.rect(screen, (255, 255, 255), self.activationArea)
    pygame.draw.rect(screen, (255, 0, 0), self.hitbox)
    screen.blit(self.transformedImg, self.hitbox.topleft)
```

This is now the display player function, I have decided to display the hitbox and activation area of the player so that I can easily see any errors while testing as otherwise the rectangles are invisible.

```
#Checks if the W key has been pressed and if it has then it deducts 1 from the player coordinates and update
#the coordinates of the player
def moveForward(self, pressed, sprintCheck):
    if pressed[pygame.K_w] and sprintCheck:
        self.y -= self.speed * self.sprintMultiplier
    if pressed[pygame.K_w]:
        self.y -= self.speed
        #print('forward')
    self.setCoords()
```

I have only taken a picture of 1 of the movement functions, however, all of the movement functions have been changes in the same way, now the procedure requires an extra boolean variable called sprintCheck which determines whether the sprint multiplier should be used on the speed of the player. I used 2 if statements, 1 for if the player is pressing the specific movement button and sprintCheck is True (Player is pressing shift button) and 1 for if the player is just pressing the specific movement button, the former applies the sprint multiplier to the speed and then changes the coordinates based on that and the latter just uses the original value for the speed of the player, the method at the end just updates the coordinates.

```
def interact(self, pressed, map):
    if pressed[pygame.K_e]:
        #Uses a function which returns a list of the walls with items and gives it an identifier
        itemWalls = map.getItemWalls()
        #Goes through each one of these walls
        for itemWall in itemWalls:
            #Checks if the player's activation area has collided with the wall with an item,
            #If the player's activation area has collided with the wall, they are close enough
            if self.activationArea.colliderect(itemWall.getRect()):
                print(itemWall.getItem())
```

This is the interact procedure so far, it takes pressed and map as parameters and then if the user presses the 'e' key on the keyboard then it sets the variable of itemWalls as the result of using the getItemWalls method on the map. This returns all of the walls with levers and hiding spaces in a list. The for loop is then used to go through each of these walls and for each wall, it checks if the activationArea of the player has collided with the wall using a method from the pygame library. If they have colided then it prints the item attribute of that wall which is another object of either type lever or hidingSpace

```
def sprintCheck(self, pressed):
    if pressed[pygame.K_LSHIFT]:
        return True
    else:
        return False
```

This is a simple function to check if the player is sprinting

```
def setCoords(self):
    self.screenCoords = (self.x, self.y)
    self.hitbox.center = (self.x, self.y)
    #Sets the coordinates of the activation area as the center of the hitbox of the player
    self.activationArea.center = self.hitbox.center
```

This function updates all of the coordinates linked to the player and sets the center of the activation area of the player as the coordinates of the center of the hitbox of the player so that the center of the player and the center of the are the same and the hitbox is at the centre of the activation area

```
def checkKeys(self, map):
    #Uses a pygame method to check if any key has been pressed and will be true if it is pressed and false if no buttons
    #on the keyboard are being pressed
    pressed = pygame.key.get_pressed()

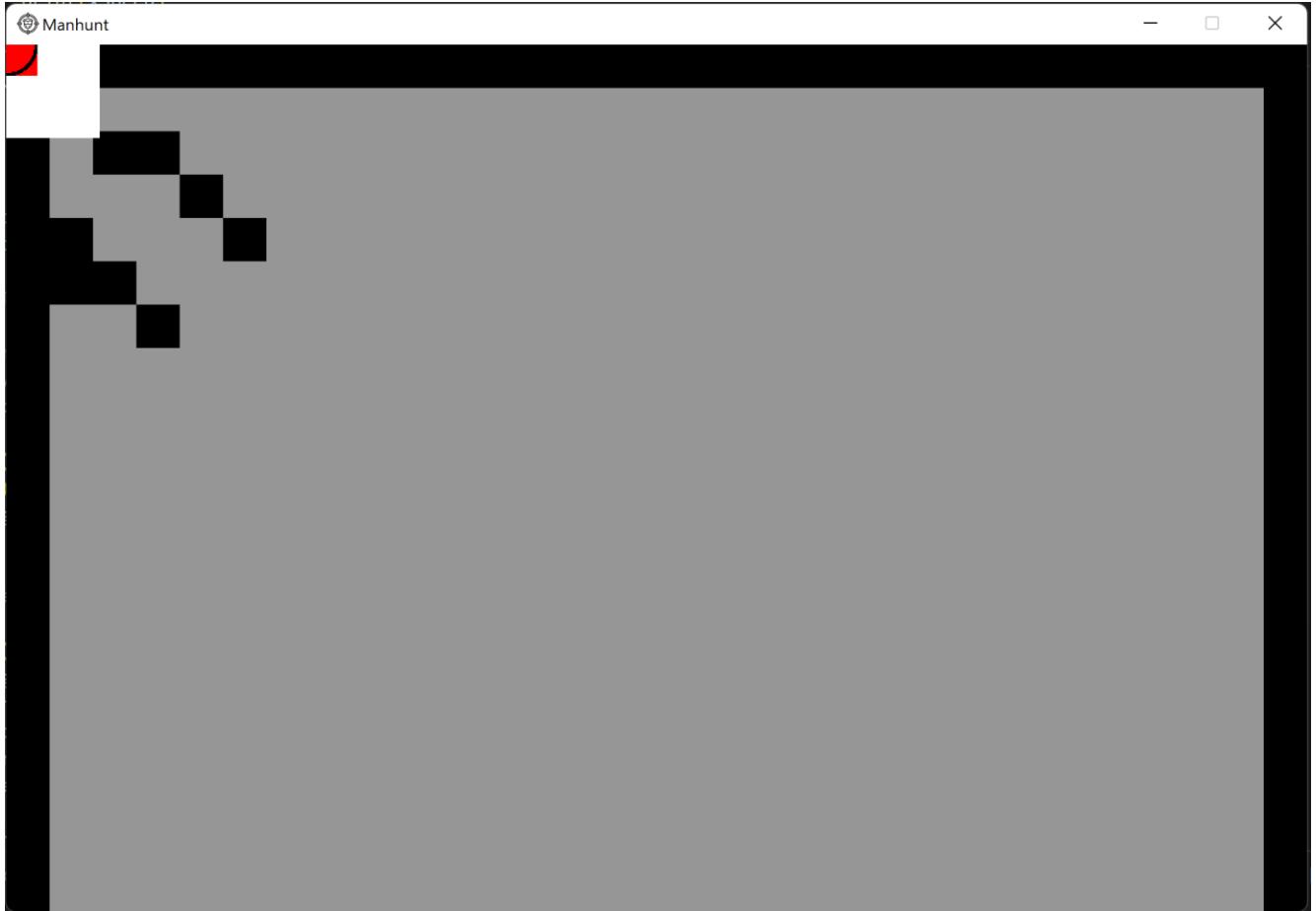
    sprintCheck = self.sprintCheck(pressed)

    #Each of these methods control an aspect of the inputs from the user which in turn controls the player
    self.moveForward(pressed, sprintCheck)
    self.moveBackward(pressed, sprintCheck)
    self.moveRight(pressed, sprintCheck)
    self.moveLeft(pressed, sprintCheck)
    self.interact(pressed, map)
```

This is the overarching method called checkKeys, this method checks all relevant inputs from the player using each method. First the pressed variable is set as the value of whatever is returned from the pygame method get_pressed, then this is passed into each method which then carries out the relevant action depending on whether it is the relevant key or not.

Name: Muqtasid Zayyan Dar

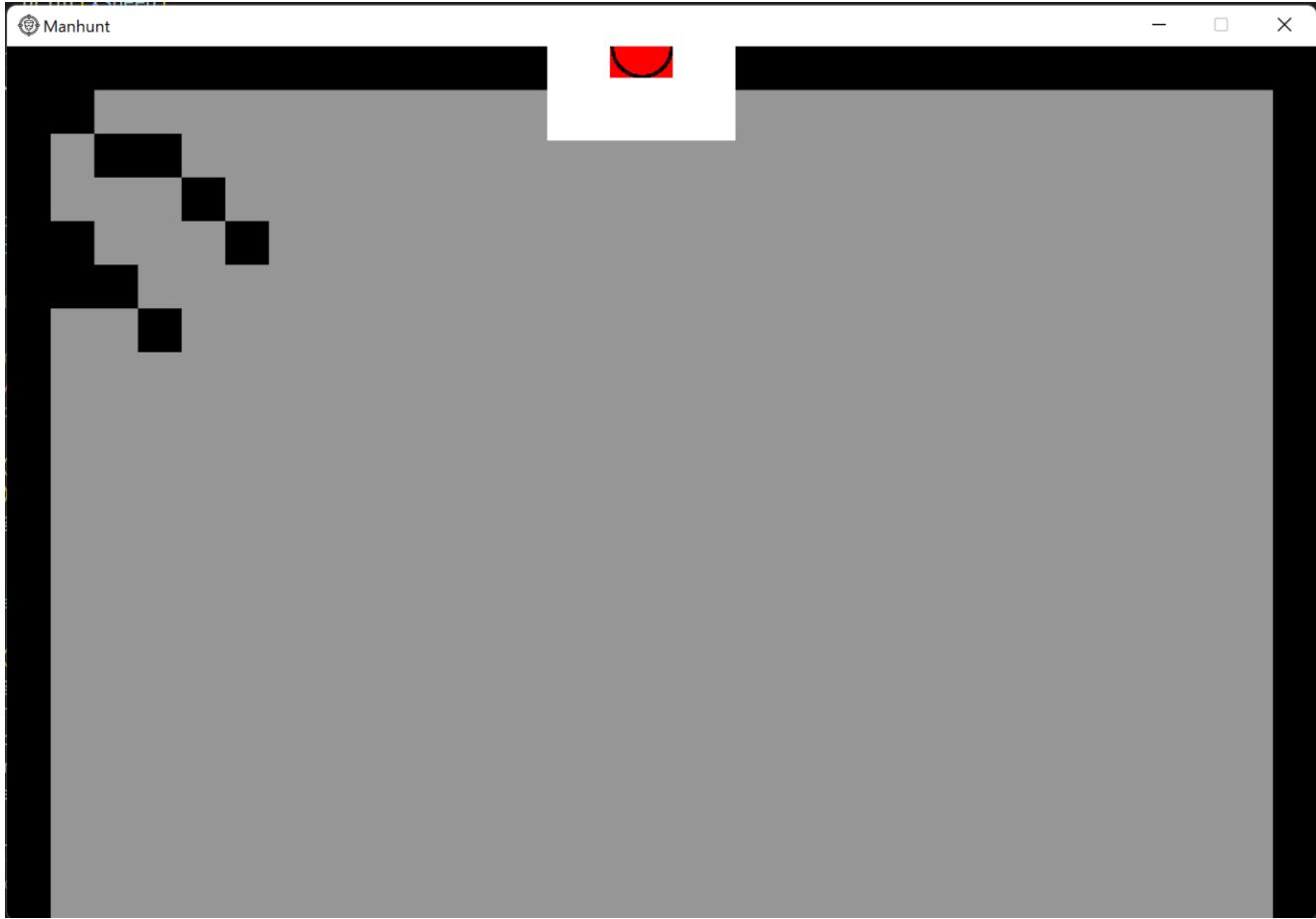
Project title: Manhunt



This is what happens when the game screen is displayed, all of the walls are in the correct locations and the player is displayed in the correct place at 0,0 on the screen, the player's activation area, hitbox and image are all displayed correctly with the image and hitbox being perfectly centered in the middle of the activation area.

Name: Muqtasid Zayyan Dar

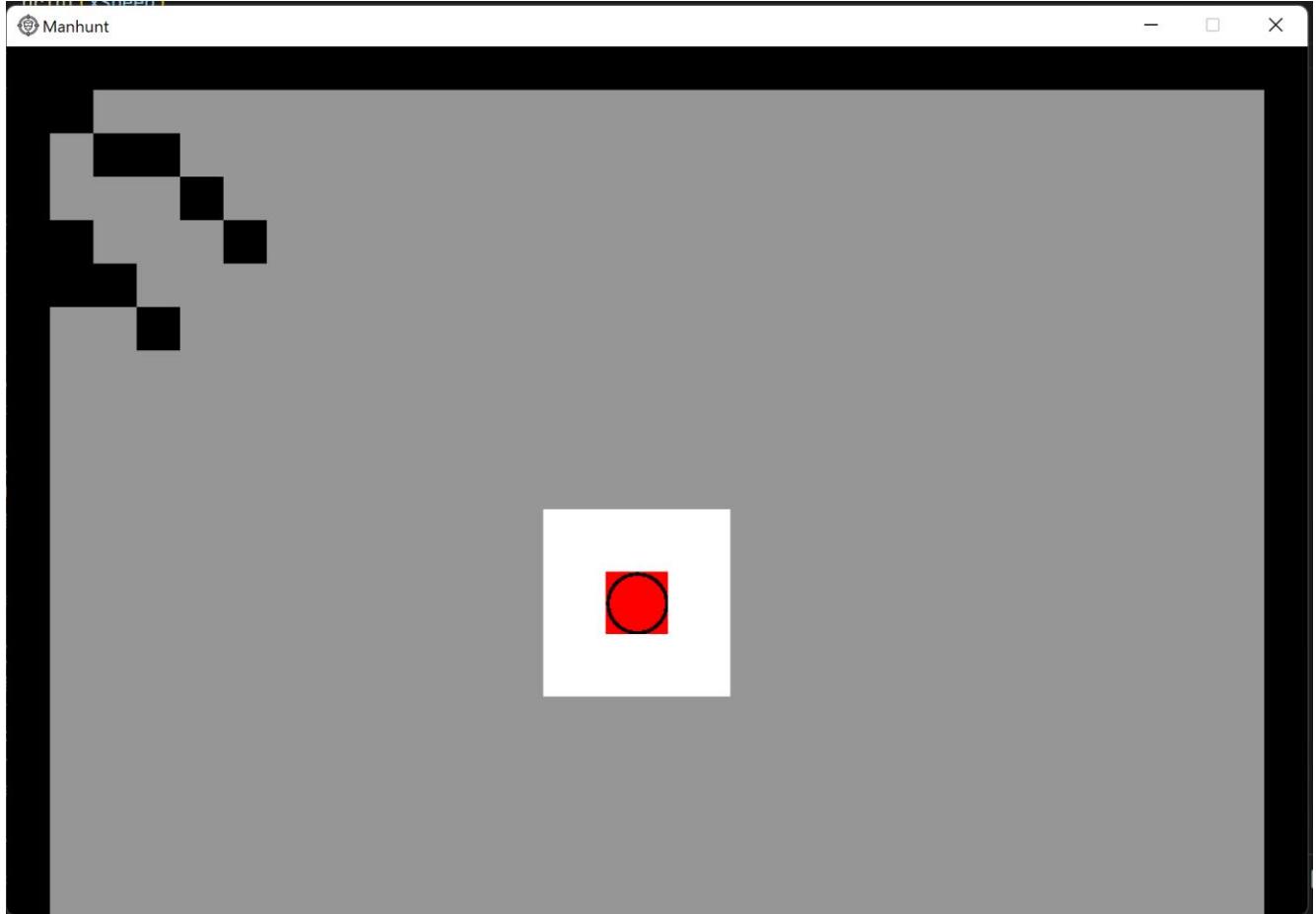
Project title: Manhunt



In this image this is what happened after I pressed the 'D' button on the keyboard and held it down, after I pressed the button once, it moved slightly to the right and stopped and while I was holding it the player, along with the other rectangles moved to the right as well which is the output I expected. Also, when I let go after holding the button then the player stopped moving.

Name: Muqtasid Zayyan Dar

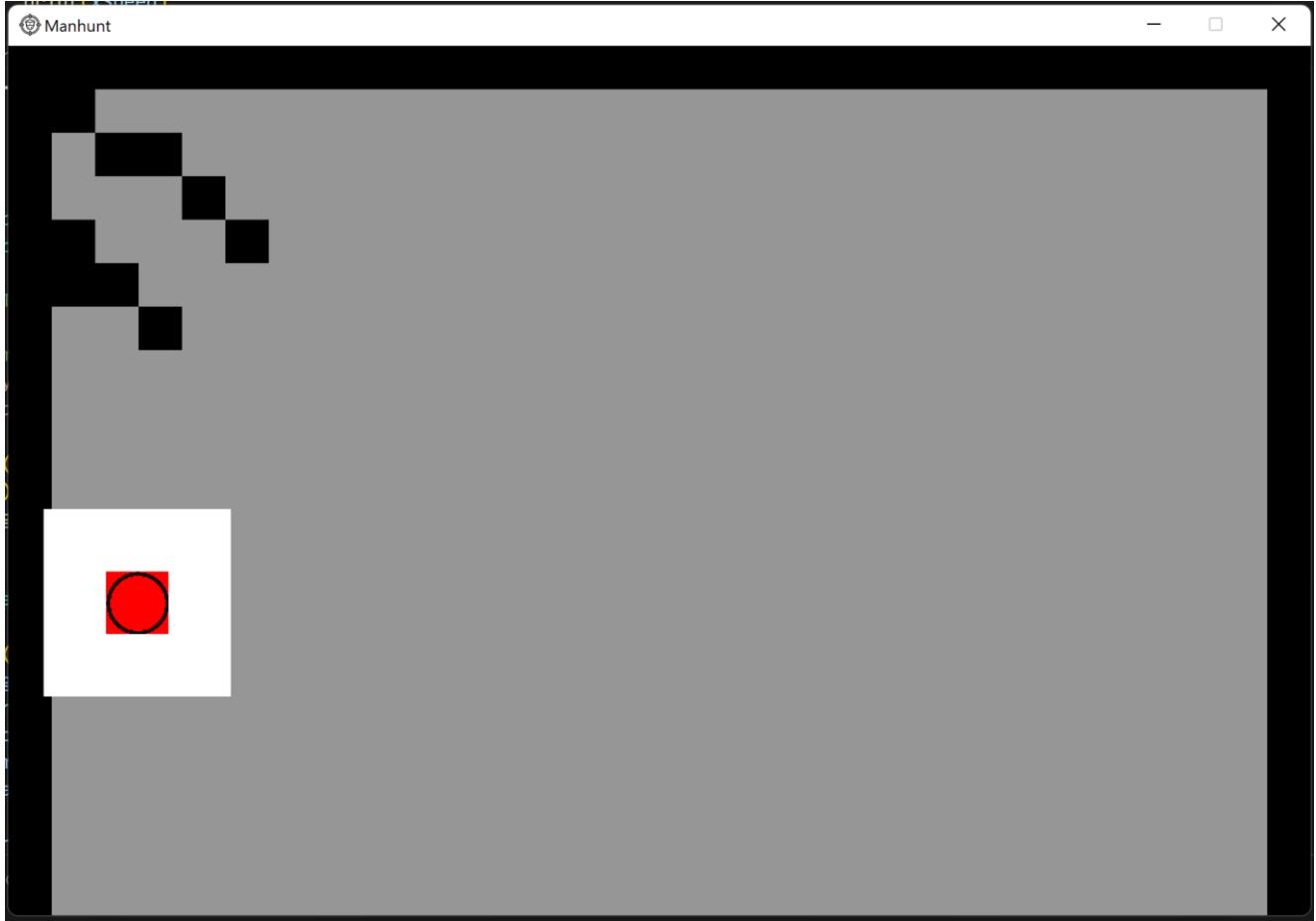
Project title: Manhunt



In this image this is what happened after I pressed the 'S' button on the keyboard and held it down, after I pressed the button once, it moved slightly down and stopped and while I was holding it the player, along with the other rectangles moved down as well which is the output I expected. Also, when I let go after holding the button then the player stopped moving.

Name: Muqtasid Zayyan Dar

Project title: Manhunt



In this image this is what happened after I pressed the 'A' button on the keyboard and held it down, after I pressed the button once, it moved slightly to the left and stopped and while I was holding it the player, along with the other rectangles moved to the left as well which is the output I expected. Also, when I let go after holding the button then the player stopped moving.



In this image this is what happened after I pressed the 'W' button on the keyboard and held it down, after I pressed the button once, it moved slightly up and stopped and while I was holding it the player, along with the other rectangles moved up as well which is the output I expected. Also, when I let go after holding the button then the player stopped moving.

```
<objects.Lever object at 0x000001CF7B704FA0>
<objects.Lever object at 0x000001CF7B704FA0>
```

When I pressed the 'E' button on the keyboard twice then this was what was printed which is the output I expected since the top left part of the player's activation area rectangle is over a wall which has a lever on it.

```
<objects.HidingSpace object at 0x000001CF7B704E50>
<objects.HidingSpace object at 0x000001CF7B704E50>
```

When I pressed 'E' button on the keyboard twice when the player was in this place then this was what was printed which was the output I expected since the top right of the player's activation area rectangle is over a wall with a hiding space.



```
<objects.HidingSpace object at 0x000001CF7B704E50>
<objects.Lever object at 0x000001CF7B704FA0>
```

When I pressed 'E' button on the keyboard once when the player was in this place then this is what was printed, although this is the output I expected, it is a problem since the player should not be able to interact with 2 objects at the same time since it could crash the program if the player tries to interact with 2 different hiding spaces at the same time. To fix this I will improve the interact method by making it check which object is closer to the player using the coordinates of the center of the player and the center of the walls, this should allow me to use a variable to keep track of which ever object is closer to the player and make sure that the player can only interact with that object.

Review:

Although the player's movement and interaction parts are now working, the fact that the player can interact with 2 objects at the same time if they are right next to each other is a problem which will be fixed in the next iteration. However, the action which should be carried out in the interaction still needs to be coded in.

Specs completed/removed:

- O1, O2, O3, O4, I1, I2, I3, I4 – These have been completed as the player can now use the WASD keys to move the player around on the screen

- Ae3 – Player now has a sprite on the screen which is a circle so has been completed
- i5, O8, P23 – Player can also now press shift button to make character move faster, therefore, these have been completed

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, P8, P9, P20, P21, O11, I6, Ae4

Specs started:

- P13, P15, O7

Version 5.3:

In this version I am going to set up some of the sound functions for the player and also fix the interaction problem where the player is able to interact with 2 different objects at the same time.

```
#This sets the sound that the player is making based on the coordinates of the player on the map
def setSound(self, map):
    coords = self.getMapCoords()
    floor = map.getObject(coords)
    self.sound = floor.getSoundLevel()

#This returns the sound level for the player
def getSound(self):
    return self.sound
```

These are the methods for getting and setting the new attribute of sound for the player, the first method which sets the sound gets the coordinates of the player on the map using the method getMapCoords() which returns the coordinates of the player in the form of a tuple. Then this coordinate is used in a different method to get the object which has the same coordinate as the player which is of type floor. Then using the getSoundLevel method which returns the sound level of the floor, self.sound is changed. The get sound level function just returns that attribute

```

def interact(self, pressed, map):
    closestDistance = 1000000
    closest = None
    if pressed[pygame.K_e]:
        #Uses a function which returns a list of the walls with items and gives it an identifier
        itemWalls = map.getItemWalls()
        #Goes through each one of these walls
        for itemWall in itemWalls:
            distance = self.playerToWallDistance(self.getCoords(), itemWall.getCoords())
            print(str(distance))
            if distance < closestDistance:
                closestDistance = distance
                closest = itemWall

        #Checks if the player's activation area has collided with the wall with an item,
        #If the player's activation area has collided with the wall, they are close enough
        if self.activationArea.colliderect(closest.getRect()):
            item = closest.getItem()
            if isinstance(item, objects.HidingSpace):
                print('Hiding Space')
            if isinstance(item, objects.Lever):
                print('lever')

```

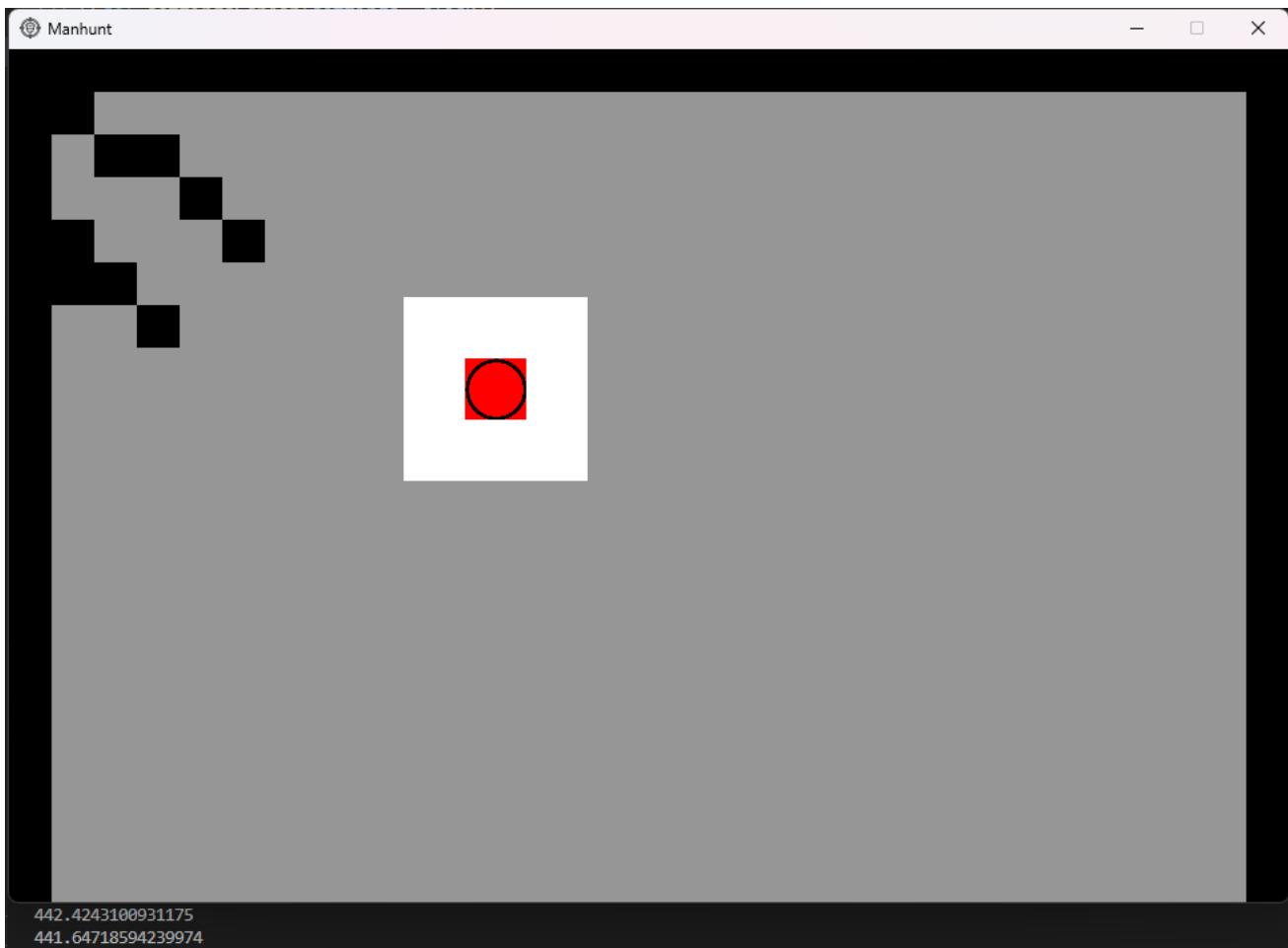
This is the interact method at the moment, I have created a new method which is an attempt at calculating the absolute distance from the player to a wall, however, this does not seem to work as can be seen in the testing later one. This method returns an integer which is then the distance between the player and that wall, if that distance, is shorter than the closest distance so far for a wall from the player then the new closest distance is that distance and closest is set to that wall object. The closestDistance has to be initialised at a high number so that it can start with the first wall it checks as having the shortest distance since the distance is based on pixels and it is not possible to be 1000000 pixels away while being on the screen. Then, for the next part, I have moved it outside the for loop so that it only checks the wall with the closest distance and therefore, the player can only interact with one item at a time. Then it checks what type of object it is and then prints out different things in each situation.

```

def playerToWallDistance(self, playerCoords, wallCoords):
    #Sets x and y coordinates of the player as variables
    xP = playerCoords[0]
    yP = playerCoords[1]
    xW = wallCoords[0]
    yW = wallCoords[1]
    #Calculates distance of player and wall from origin using method from physics class
    playerDistance = self.pythagoras(xP, yP)
    #print(playerDistance)
    wallDistance = self.pythagoras(xW, yW)
    #print(wallDistance)
    distance = abs(playerDistance - wallDistance)
    return distance

```

This is the attempt at calculating the distance between the player and the walls, it is a method which takes the coordinates of the player and the wall coordinates as the parameters and then sets each x and y value for each set of coordinates with different identifiers 'P' is for Player and 'W' is for Wall. Then the direct distance from the origin to the player and the origin to the wall is calculated and then the wall distance is taken away from the player distance and changed to an absolute value as negative numbers could cause problems when comparing the shortest distances because, one wall may be further away but in the opposite direction and one may be closer but in the positive direction and the closest distance comparison may still choose the one in the opposite direction since it is negative. Then this value is returned. However, this does not calculate the correct value from the player as will be seen in the testing below. The reason is for the logic behind this because all values are being calculated relative to the origin and this causes issues, as the same wall will always be the closest one to the origin.. What I need to do is calculate the values relative to the player.



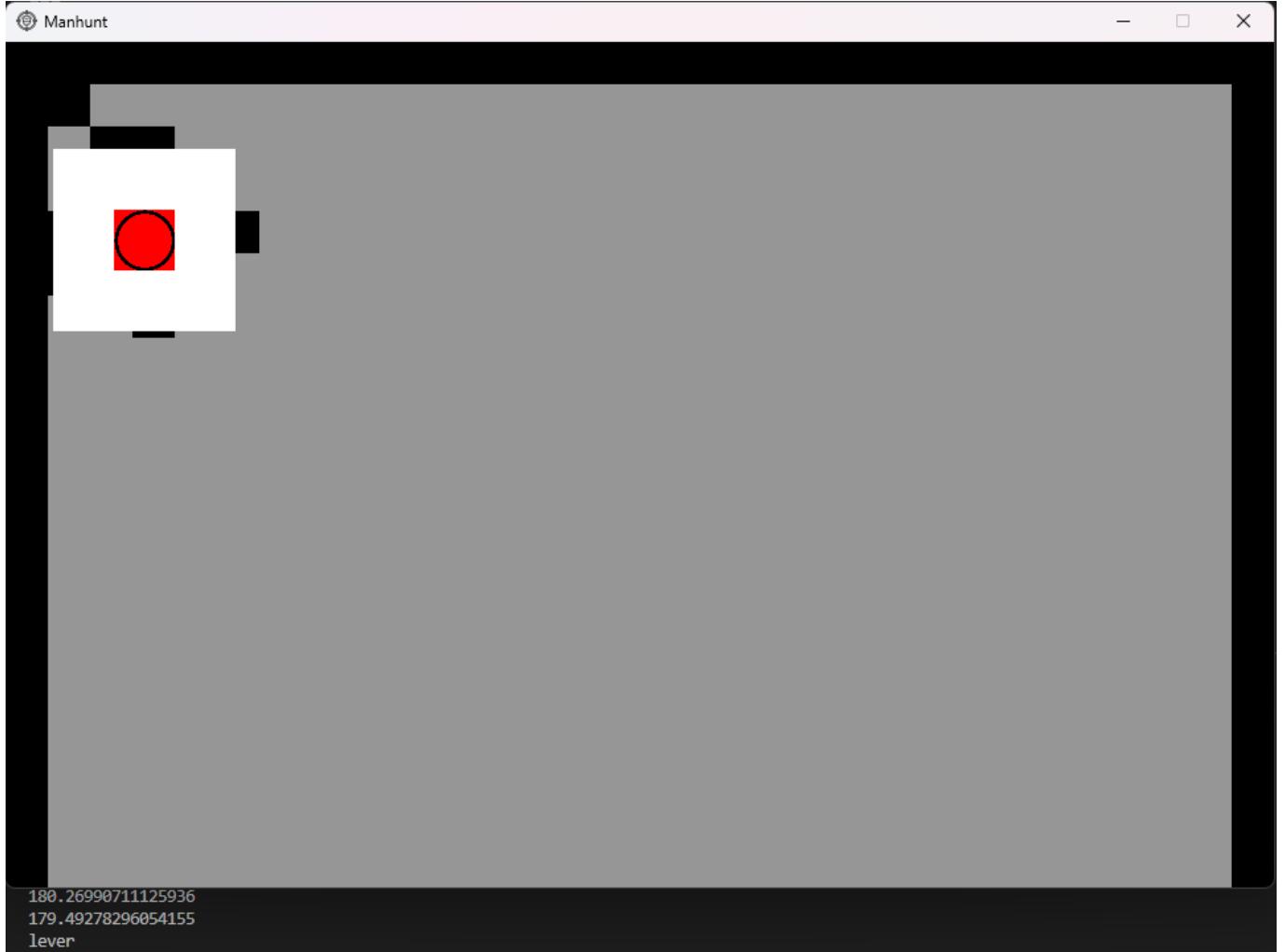
In this picture, I pressed 'E' and did not expect either 'lever' or 'Hiding Space' to be printed, the distances at the bottom of the hiding space and the lever seem to be correct in this situation.

Name: Muqtasid Zayyan Dar

Project title: Manhunt



In this picture, I pressed 'E' and expected 'lever' to be printed which was output, the distances at the bottom of the hiding space and the lever seem to be correct in this situation as the lever is closer than the hiding space



In this picture, I pressed 'E' and expected 'lever' to be printed which was output, the distances at the bottom of the hiding space and the lever seem to be correct in this situation as the lever is closer than the hiding space. This was a similar case that was causing a problem before.



In this picture, I pressed 'E' and expected 'Hiding Space' to be printed, however, nothing was printed and the numbers at the bottom display the problem. I have done my calculations incorrectly and the hiding space is still being calculated as further away than the lever and so nothing was printed. I have decided that this is an unnecessary addition to my game and it would be inefficient to spend time trying to figure this out so I will instead make sure that the levers and hiding spaces are far away enough in the actual map that this cannot occur. This will be done by making sure that the closest distance between 2 interactable objects is greater than the distance of the activation area of the player since this will mean that it would be impossible for the player to interact with more than 1 interactable objects.

Review:

Although initially I believed that it would be important to solve this problem by adding some more calculation for the player to do, I now think that it would be more efficient to just ensure that this doesn't happen by making the map in such a way that 2 interactable objects are never close enough for the player to be able to interact with both at the same time.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, P8, P9, P20, P21, O11, I6, Ae4, P13, P15, O7

Specs started:

- None

Version 5.4:

In this version I am going to finish off the interaction part of the player for the levers and the hiding spaces so that the player can easily interact with either and it carries out the relevant action based on whether it is a hiding space or a lever

```
#This is the current state of whether the player is hiding or not
self.hiding = False
#These are the attributes for the levers
self.maxLevers = None
self.activatedLevers = 0
```

These are the new attributes I have added. The first attribute, self.hiding, is a boolean value which will be changed and represents the hiding state of the player, false if the player is not hiding and true if the player is hiding. Then there are the next 2 attributes which are for the levers, the self.maxLevers is set to None as a method later on will determine what it is and then the next attribute represents the number of levers that the player has activated and is set to 0 as at the start the player has not activated any levers.

```
#This procedure carries out the interaction function for the player
def interact(self, pressed, map):
    if pressed[pygame.K_e]:
        if self.hiding == False:
            #Uses a function which returns a list of the walls with items and gives it an identifier
            itemWalls = map.getItemWalls()
            #Goes through each one of these walls
            for itemWall in itemWalls:

                #Checks if the player's activation area has collided with the wall with an item,
                #If the player's activation area has collided with the wall, they are close enough
                if self.activationArea.colliderect(itemWall.getRect()):
                    #item is the wall which also contains either a hiding space or lever
                    item = itemWall.getItem()
                    self.hideCheck(item)
                    self.leverCheck(item, map)

            else:
                self.setHiding(False)
```

This is the new version of the interact method, I have gotten rid of the distance estimation stuff and instead focused on refining it. It works in a similar way as the first iteration of this method, however, now the player is only able to interact with things if they are not hiding (self.hiding = False) and if they press the 'E' button while they are hiding (self.hiding = True) then it just unhides them using a method

to set hiding as false again – this method will be shown later on. Also, now if the player's activation area is colliding with either a lever or a hidingSpace then it sets a new variable called item as that object and then runs 2 different methods which check if it is their respective object and then carry something out. In the leverCheck method, I have passed item and map, however, map is not needed and will be removed in future iterations.

```
def leverCheck(self, item, map):
    #Checks if the item is a lever
    if isinstance(item, objects.Lever):
        #Checks if the lever has already been activated
        if item.getActivated() == False:
            self.activateLever()
            #Activates that specific lever so it cannot be activated again
            item.activate()
            print('activated lever')

    else:
        print('LEVER HAS ALREADY BEEN ACTIVATED')
```

This is the leverCheck method and as stated above, it has a parameter (map) which is not needed and will be removed later on. This method uses a built in method to check if the item passed into the method is a lever object and if it is then it checks if the lever has already been activated so that the player cannot activate a lever multiple times. If that lever has not been activated then the self.activateLever() method is run which will be talked about later and then the activate() method is run on the lever to set its activation from false to true, the print statement is used to test whether this part runs correctly. Furthermore, if that specific lever has already been activated then a statement will be printed out (for testing) so that I can check that it only works when the lever has not been activated.

```
#This procedure checks if the object is a hiding space
def hideCheck(self, item):
    #Checks if the item is a hidingspace
    if isinstance(item, objects.HidingSpace):
        self.setHiding(True)
        print('Hiding Space')
```

This is the hideCheck method and it is similar to the leverCheck function as it checks if the object passed (item) is of type HidingSpace, and if this is true then it runs the setHiding method and passes true to it – This method will also be explained later

```
#This method checks if the W, A, S, D, E, or LEFT SHIFT buttons are pressed and then carries out various actions depending
#on the button that is pressed
def checkKeys(self, map):
    #Uses a pygame method to check if any key has been pressed and will be true if it is pressed and false if no buttons
    #on the keyboard are being pressed
    pressed = pygame.key.get_pressed()
    hiding = self.getHiding()
    sprintCheck = self.sprintCheck(pressed)

    #Each of these methods control an aspect of the inputs from the user which in turn controls the player
    if hiding == False:
        self.moveForward(pressed, sprintCheck)
        self.moveBackward(pressed, sprintCheck)
        self.moveRight(pressed, sprintCheck)
        self.moveLeft(pressed, sprintCheck)
    self.interact(pressed, map)
```

This is the new version of the checkKeys method, it has not been changed much, however, the getHiding method is used to check if the player is hiding and this is given the identifier ‘hiding’, then then if statement checks whether this is false which means the player is not hiding and if the player is not hiding then the other movement methods can be carried out. However, if the player is hiding then they are not able to move (as hiding would be True), this is to ensure that the player cannot move around the map while they are hiding as this would be a bug if the player could hide and move around the map at the same time. However, the interact method is placed outside of this if statement as the player needs to press ‘E’ to be able to leave the hiding space once they have gone inside it.

```
#This returns the value for the number of activated levers
def getActivatedLevers(self):
    return self.activatedLevers

#This function adds 1 to the number of activated levers if not all of the levers have been activated
def activateLever(self):
    self.activatedLevers += 1
    print(self.activatedLevers)

#This returns the max number of levers that are in the game
def getMaxLevers(self):
    return self.maxLevers

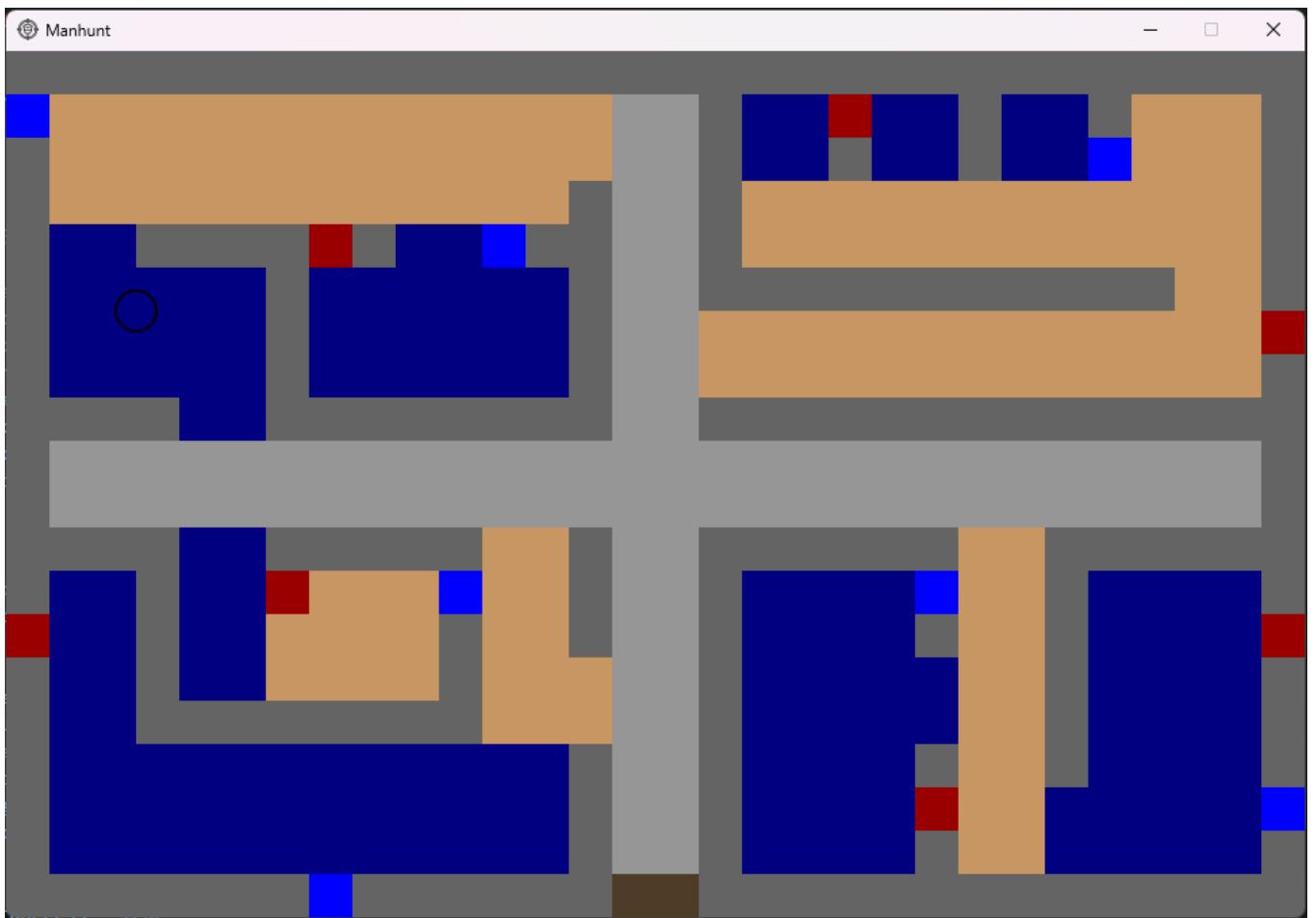
#This procedure gets a list of all of the levers from the map class and then sets the number of max levers as the length of that list
def setMaxLevers(self, map):
    levers = map.getWalls('lever')
    self.maxLevers = len(levers)
```

These are the methods for getting and setting the attributes: activatedLevers and also MaxLevers. The get functions for both attributes just return their respective attributes. The activateLever method increments the number of activatedLevers by 1 and is used when the player activates a lever. The print statement in that method is for testing purposes so that I can see that the attribute is incremented. The last method which is the setMaxLevers method, takes the parameter map which is the map object. Then a method is used on this object which returns a list of all of the levers in the map as a list and then gives it the identifier of levers. Then the maxLevers attribute is set as the length of this list as this would be all the levers that the player can activate in the map.

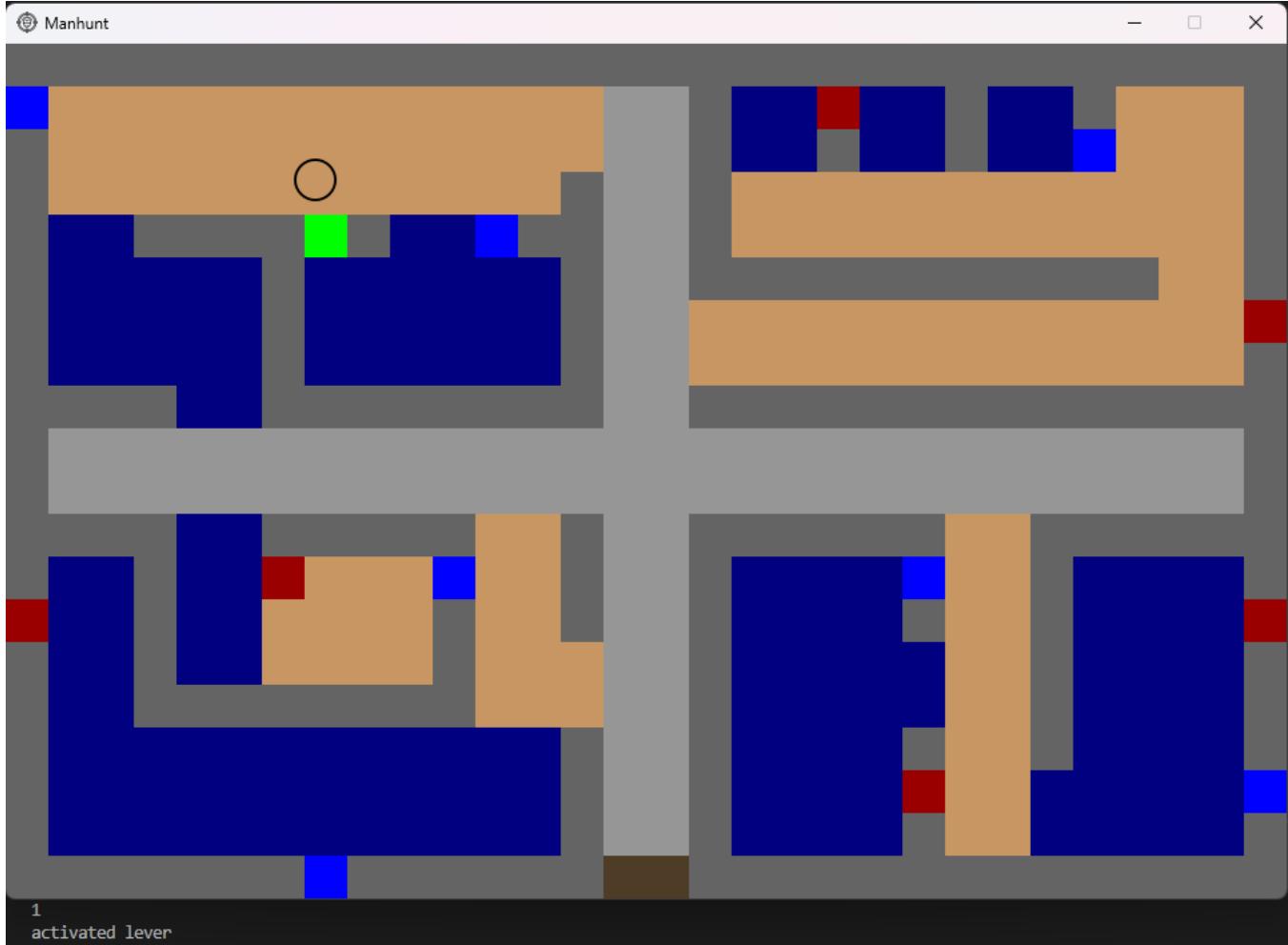
```
#This returns a boolean value of the attribute for if the player is hiding or not
def getHiding(self):
    return self.hiding

#This sets the value of hiding pabased on the parameter that is passed which has to be a boolean value
def setHiding(self, hiding):
    if hiding == True or hiding == False:
        self.hiding = hiding
    else:
        print('HIDING IS NOT BOOLEAN VALUE')
```

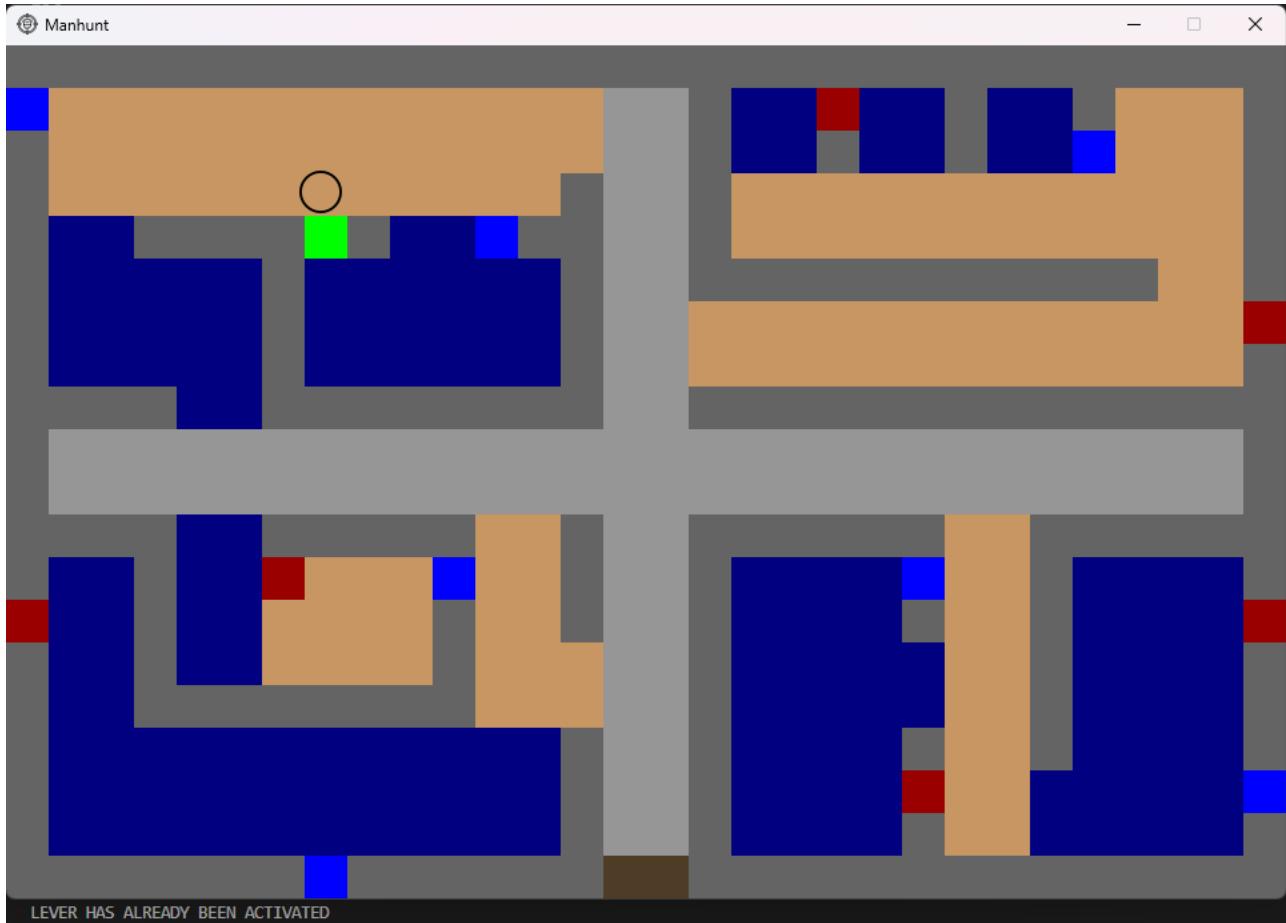
These final 2 methods are the get and set methods for the hiding attribute. The getHiding method just returns self.hiding which is a boolean value and then the setHiding method takes a value and then sets self.hiding as that value, however, the data passed into this method is validated to ensure that the parameter passed is a boolean value.



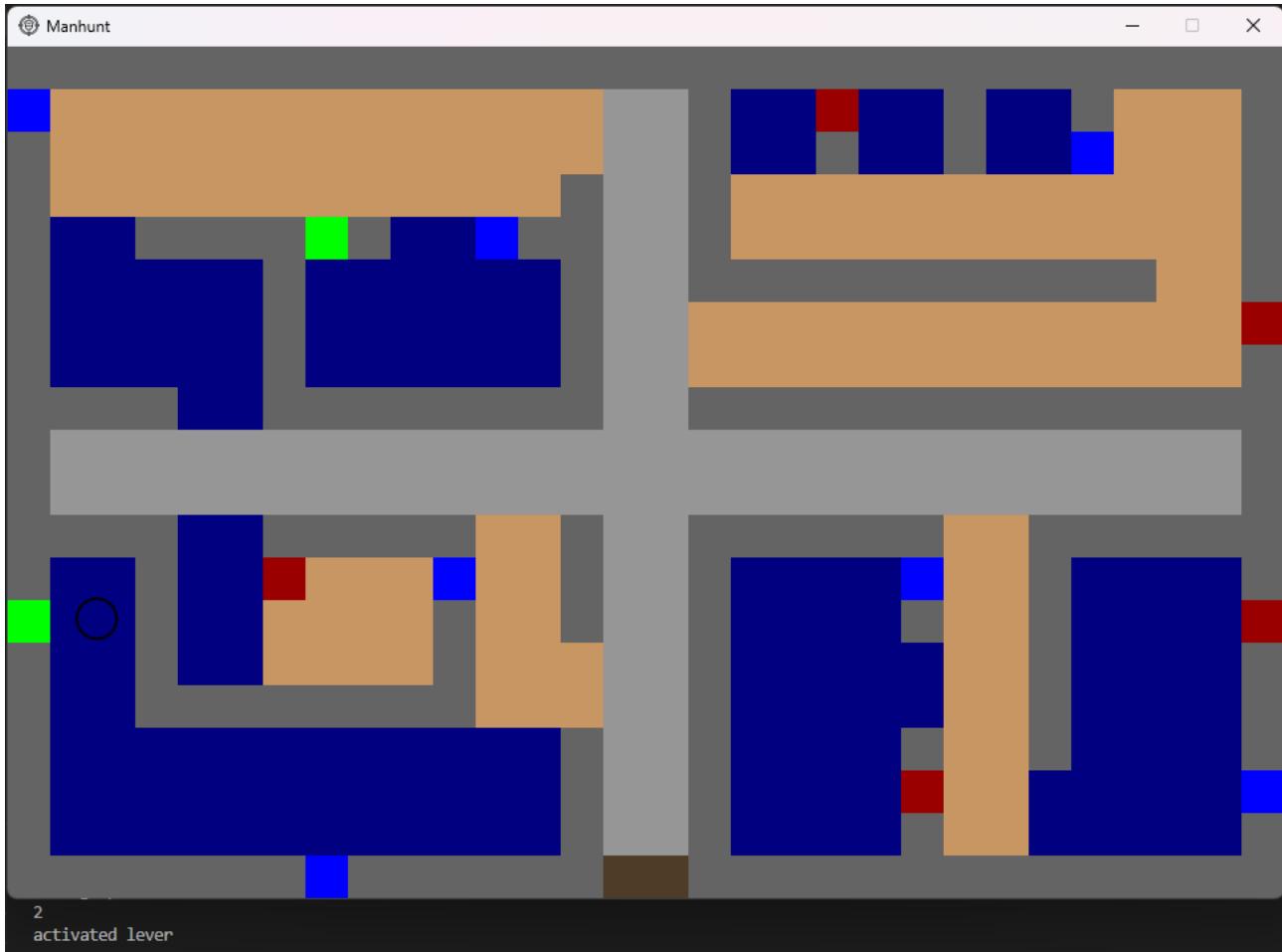
When the player first spawns into the map, then I expect that none of the levers should be activated. This is also what happened when I tested it.



This is a valid test, I moved the player close to the lever (so that the lever is within the activation area of the player) then I pressed the 'E' key on the keyboard. I expected the number of activated levers printed to be 1 as only 1 is activated and then 'activated lever' to also be printed out. I also expected that the lever that I activated would change colours to signify it has been activated. When I carried out the test, then the number of activated levers printed out was 1 and 'activated lever' was also printed out. Furthermore, the colour of that lever also changed from red to green which shows it has been activated to the user. The success criteria for this test was completely met.



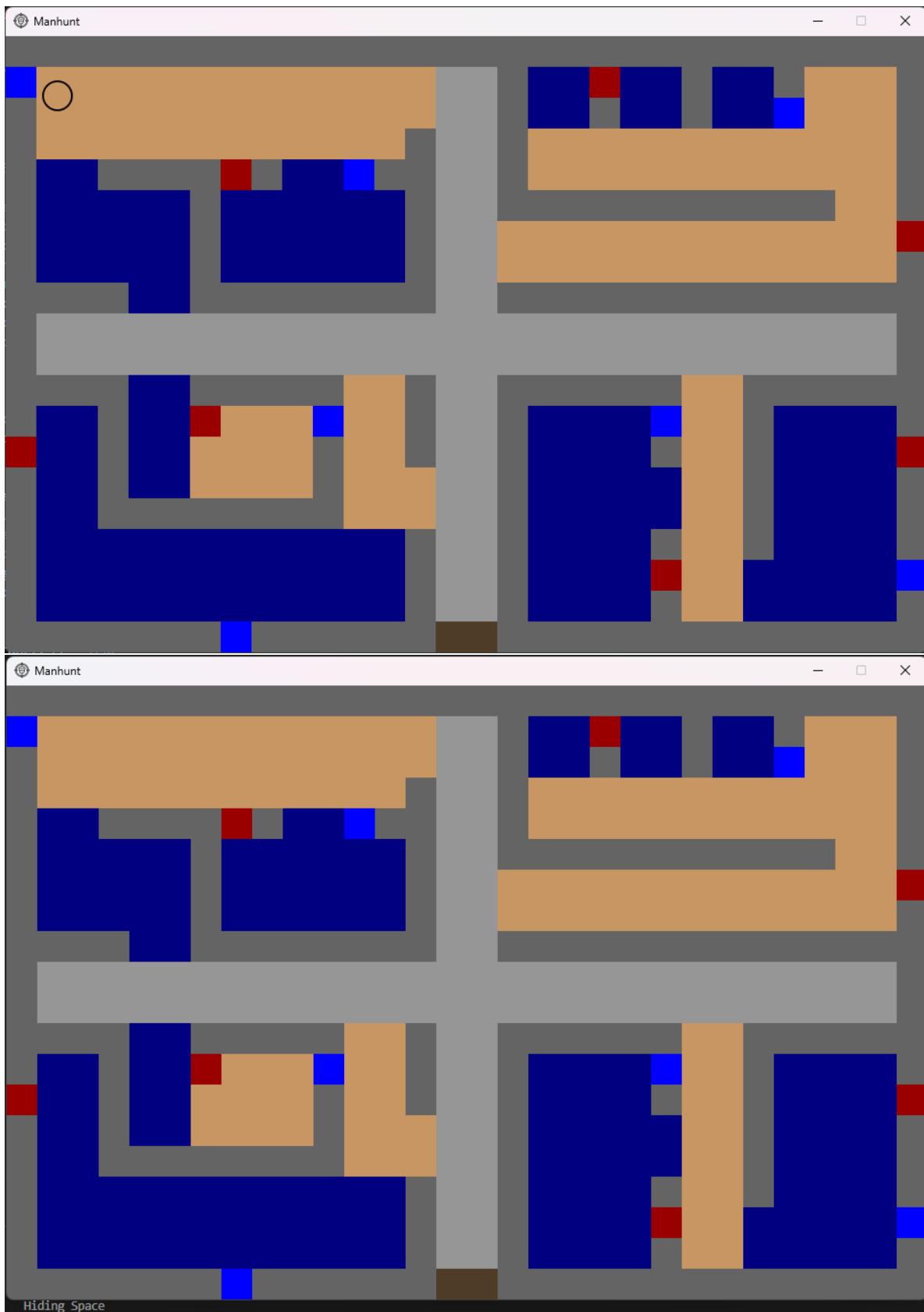
This is an invalid test, as the lever has already been activated, the player should not be able to activate the same lever more than once. When I press the 'E' key in this position, the expected output is that nothing should happen and the number of activated levers should not be incremented. Also, the colour of the lever should not change when 'E' is pressed. When I carried out this test, the only thing that was printed out was that 'LEVER HAS BEEN ACTIVATED', this is what I expected as this print statement means that the number of activated levers has not changed. Furthermore, the colour of the lever on the screen did not change from green back to red or anything similar. Therefore, the success criteria for this test was completely met.



This is a valid test, I moved the player close to a second lever which has not been activated and then pressed the 'E' key. In this situation, the same output should happen as the first time, except the number of activated levers printed out should be incremented by 1. When I carried out this test, then the number of activated levers printed out was 2 which is correct as I have only activated 2 levers. The rest of the outputs are the same as the first time with 'activated lever' being printed out and the colour of that lever changing from red to green to show it has been activated. Therefore, the success criteria for this test was also completely met.

Name: Muqtasid Zayyan Dar

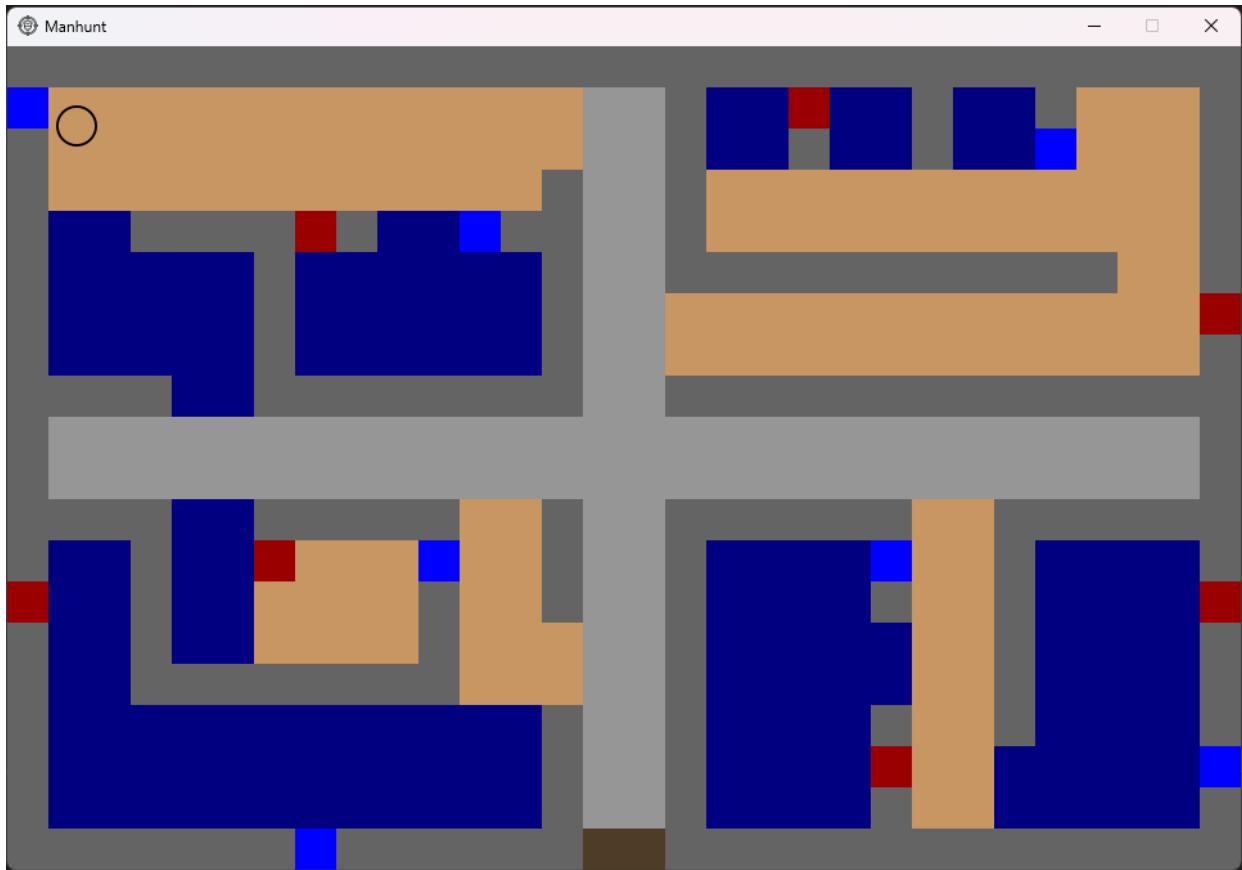
Project title: Manhunt



This is a valid test, I moved the player next to a hiding space and then pressed the 'E' key to interact with it. The expected output is that the player disappears from the screen. This is what happened, and since 'hiding space' was printed out, it means that the right actions were carried out for the hiding space. Therefore, the success criteria has been met.

The next test continues while the player is hiding and is an invalid test. When I press any of the movement keys - 'W', 'A', 'S', 'D', or 'Shift' then the expected output is that the player should not change positions when the player comes out of hiding.

Another thing that is being tested is that when the user presses the 'E' key while hiding then the expected output is that the player should reappear on the screen and the player should be able to use the movement keys and interact with other objects.



Above is the result after testing the 2 tests above, after pressing each of the movement keys, the player has not moved at all and is in the same position as before. Also, when 'E' is pressed then the player reappears on the screen.

Review:

In this iteration, I learned about the `isinstance` function which can be used in order to identify the data type for some data. This was useful as it helped when I needed to figure out when an object was a lever or a hiding space so the player was able to carry out the correct action. I decided to test these methods

in the map which I created previously to be used in the end game as I believed it would show a more realistic result of what happens when the player interacts with the environment.

Specs completed/removed:

- I6, P13, O7 – As the player is able to now hide in hiding spaces when the player presses the E key and the character disappears from the screen when this happens, these have been completed
- P15 – When the player interacts with a lever then its colour changes form red to green so this has also been completed

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, O5, P8, P9, P20, P21, O11, Ae4

Specs started:

- None

Version 5.5:

In this version I am going to be creating the collisions for the player and also the win condition as the player needs to be able to win the game, the player can win after all of the levers have been activated and the player collides with the door. I will also refine some of the previous components of the program which I have completed previously

```
self.forward = False  
self.backward = False  
self.left = False  
self.right = False
```

These are the new boolean attributes I have added to be used in collisions for the player

```
#Checks if the W key has been pressed and if it has then it deducts 1 from the player coordinates and update
#the coordinates of the player
def moveForward(self, pressed, sprintCheck):
    if pressed[pygame.K_w] and sprintCheck:
        self.y -= self.speed * self.sprintMultiplier
        self.setMovement(True, 'F')
    elif pressed[pygame.K_w]:
        self.y -= self.speed
        self.setMovement(True, 'F')
    else:
        self.setMovement(False, 'F')
    self.setCoords()
```

I have modified each of the movement methods in the same way which uses a method which will be explained below to change the values of the new attributes based on whether the player is moving or not and in which direction

```
#This method checks if the W, A, S, D, E, or LEFT SHIFT buttons are pressed and then carries out various actions depending
#on the button that is pressed
def checkKeys(self, map):
    #Uses a pygame method to check if any key has been pressed and will be true if it is pressed and false if no buttons
    #on the keyboard are being pressed
    pressed = pygame.key.get_pressed()
    hiding = self.getHiding()
    sprintCheck = self.sprintCheck(pressed)
    self.checkWin(map)

    #Each of these methods control an aspect of the inputs from the user which in turn controls the player
    if hiding == False:
        self.moveForward(pressed, sprintCheck)
        self.moveBackward(pressed, sprintCheck)
        self.moveRight(pressed, sprintCheck)
        self.moveLeft(pressed, sprintCheck)
        self.interact(pressed, map)
```

This is the new version of the checkKeys procedure, I have only added a statement which invokes the checkWin procedure which is passed map and will be talked about later on

```
#This function sets the movement directions as a boolean value based on the parameters passed
def setMovement(self, value, direction = None):
    #Validates that the value passed is a boolean value
    if value != True and value != False:
        print('THIS VALUE IS INVALID, MUST BE BOOLEAN')
    #If data is validated then attribute can be modified based on the direction passed
    else:
        #Each value can be changed based on the direction, however, 2 cannot be changed at the same time
        #in this function
        if direction != None:
            if direction == 'F':
                self.forward = value
            elif direction == 'B':
                self.backward = value
            elif direction == 'L':
                self.left = value
            elif direction == 'R':
                self.right = value
        else:
            self.forward = value
            self.backward = value
            self.left = value
            self.right = value
```

This is the set movement procedure, based on a boolean value and a string that is passed into the procedure, it will set a certain directional attribute of the player (forward, backwards, left or right) as true or false depending on that boolean value. However, in future iterations I will most likely remove this as it takes up more space than if I just set the values in the separate movement functions at that time.

```
#This function checks if the player has activated all of the levers and then returns a boolean value based on that
def checkWin(self, map):
    #function returns the door objects in a list
    doors = map.getDoor()
    print(self.getActivatedLevers())
    print(self.getMaxLevers())
    #checks if the player has activated all of the levers
    if self.getActivatedLevers() == self.getMaxLevers():
        #As there are multiple doors, this loop goes through each door
        for door in doors:
            door.activate()
            #Checks if each door has been collided with and carries out action based on that
            if self.hitbox.colliderect(door.getRect()):
                print('WIN')
```

This is the checkWin procedure, it uses a method on the parameter map to get all of the doors and then it checks if the number of activated levers is equal to the max number of levers using their respective methods as this would mean that all of the levers have been activated if this is true. If all of the levers have been activated, then a for loop is used to go through each of the doors in that list from the map and each door is activated. Then an if statement is used which uses a built in pygame method which checks if the hitbox of the player has collided with the door which means the player has escaped, if this happens then something is printed out (for testing) to show that it works, however, later on it will be used to return a value which will take the player to the win screen.

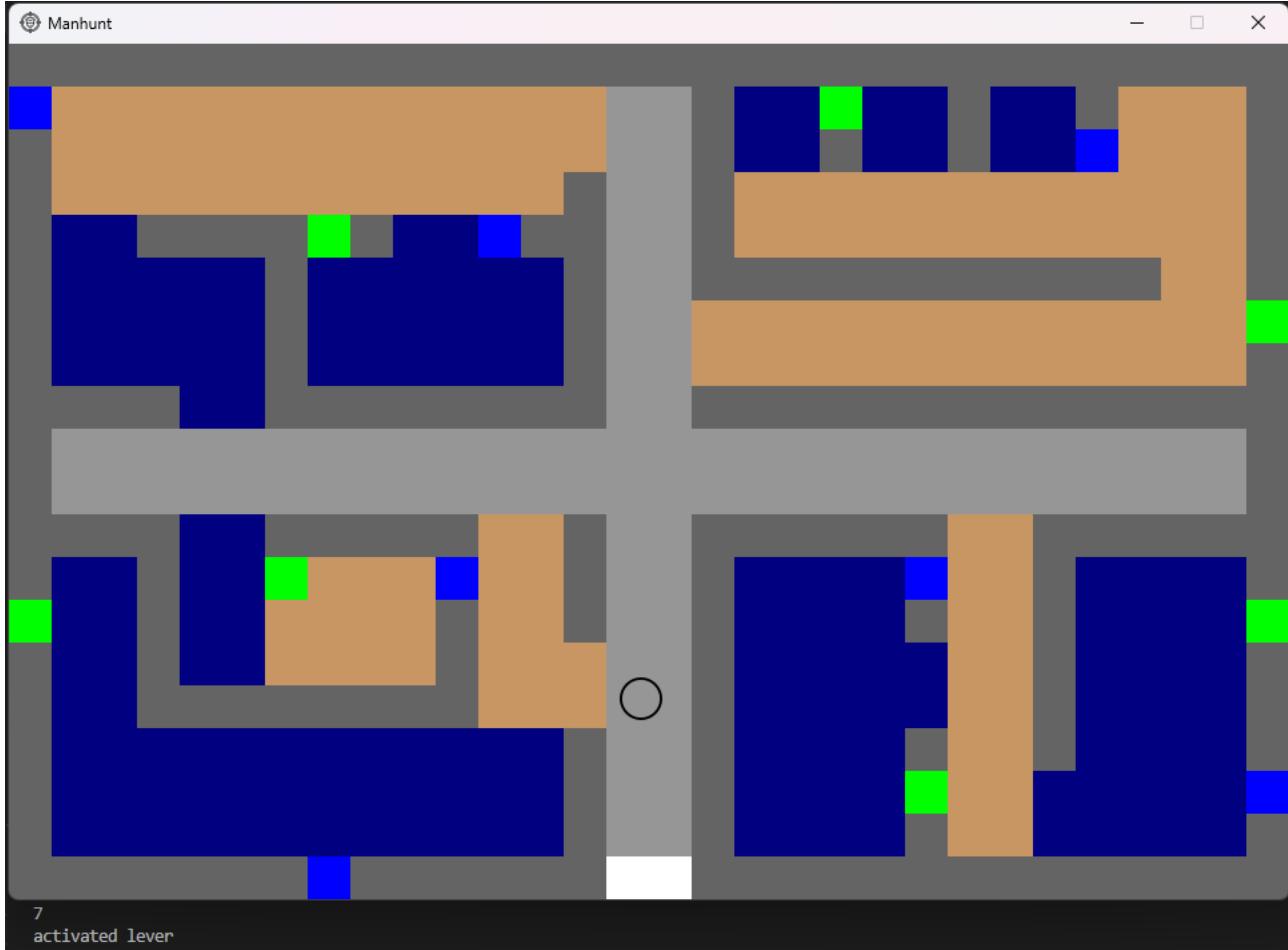
```
#This is the function which checks for collisions of the player with walls and then causes the player to bounce off of it
def checkCollision(self, map):
    #This function returns all of the walls for the map in a list
    walls = map.getWalls()
    collisionTolerance = 15
    #This is how many pixels the player will bounce off the wall when collision occurs
    collisionBounce = 10
    player = self.hitbox
    #This loop goes through each wall in the list above
    for wall in walls:
        #Gets the rectangle of the wall
        rect = wall.getRect()
        #Checks if the rectangle of the player has collided with that rectangle (for the wall)
        if self.hitbox.colliderect(rect):
            print('COLLIDING')
            #Each of these if statements check which direction the player is currently moving
            #and based on this decision, the direction the player bounces back is determined
            if self.backward == True:
                print('BOTTOM')
                self.y -= collisionBounce

            if self.forward == True:
                print('TOP')
                self.y += collisionBounce

            if self.right == True:
                print('RIGHT')
                self.x -= collisionBounce

            if self.left == True:
                print('LEFT')
                self.x += collisionBounce
```

This is the checkCollision procedure which takes map as a parameter. The collisionTolerance variable and player variable can be ignored as they will be removed later on since they are not needed. It functions in the same way to the function which I created in version 4.1. However, it goes through every single wall in the map in order to check if the player's hitbox has collided with any walls in the map.



This is a valid test, for this test, I activated all of the levers in the map which means the number of activated levers must be 7. For this test, the expected output was that the doors should be unlocked

Review:

In this iteration, I used what I had learned to implement collisions in 4.1 and implemented it here in a slightly different format so that the player could not walk through the walls in the map. Furthermore, I also used this collision technique in order to be able to check when the player collides with the door and all of the levers have been activated for the player to win. Now that all the other parts of the player have been added, the final part for the player is the field of view. This will be coded in the next iteration

Specs completed/removed:

- P17 – I have slightly altered this criterion and now it is only about the player completing the objective and that the exit door will open when that happens as I believe that the extra text is unnecessary. Therefore, this is now complete as the exit door opens once the player has activated all of the levers.
- O5 – The player is now able to collide with the walls and doesn't go through them, therefore, this has been completed

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, P8, P9, P20, P21, O11, Ae4

Specs started:

- O9

Version 5.6:

In this version I am going to be creating the player's FOV so that the vision of the player is realistic and the player cannot see through walls or other solid things. Furthermore, I am also going to finish off the sound functions for the player

```
#This creates the projectiles around the player which will be used for the field of view
def createProjectiles(self):
    #This is the length of each side of the projectile in pixels which will be passed into the projectiles class
    length = 32

    #As there was no simpler way to create a loop as the projectiles need a specific speed, each one had to be created individually
    projectile1 = physics.SightProjectile(self.getHitbox().topleft, 0, 5, length)
    projectile2 = physics.SightProjectile(self.getHitbox().topleft, 0, -5, length)
    projectile3 = physics.SightProjectile(self.getHitbox().topleft, 5, 0, length)
    projectile4 = physics.SightProjectile(self.getHitbox().topleft, -5, 0, length)
    projectile5 = physics.SightProjectile(self.getHitbox().topleft, -5, -5, length)
    projectile6 = physics.SightProjectile(self.getHitbox().topleft, -5, 5, length)
    projectile7 = physics.SightProjectile(self.getHitbox().topleft, 5, 5, length)
    projectile8 = physics.SightProjectile(self.getHitbox().topleft, 5, -5, length)

    #As the projectiles were created as individual variables, they must be appended to the projectiles list individually as well
    self.sightProjectiles.append(projectile1)
    self.sightProjectiles.append(projectile2)
    self.sightProjectiles.append(projectile3)
    self.sightProjectiles.append(projectile4)
    self.sightProjectiles.append(projectile5)
    self.sightProjectiles.append(projectile6)
    self.sightProjectiles.append(projectile7)
    self.sightProjectiles.append(projectile8)
```

This is the `createProjectiles` method which creates all of the projectile in all of the different directions which will be used for the player to be able to see. First the variable `length` is set as 32 which will be the length of the sides of the projectiles when passed to the `sightProjectiles` class. Then I had to make 8 different variables for 8 different projectiles. This is because there was no repeating pattern in the speeds which each of the projectiles had to go in so I was not able to think of an iterative solution to this problem which could produce the same outcome. Each projectile is created with a certain speed in the x or y direction, I have decided on it being 5 (in the positive or negative x or y direction). The first 4 projectiles are created with a speed in only 1 axial direction so they originate directly from the player and move parallel to the axis. The last 4 projectiles have both an x and y speed originate and travel diagonally from the player in different directions. Then I had to append each of these projectiles to the new `sightProjectiles` attribute for the player which are all of the sight projectiles in one list.

```

def fov(self, map, screen):
    #This variable holds a 1D list of all of the objects in the map
    allObjects = map.getAllObjects()
    #This for loop goes through each of the objects in the list above
    for object in allObjects:
        #This sets the visibility of each object as False so that it appears black on the screen
        object.setVisible(False)

    #This checks if the projectiles have been made yet as they are appended to the list straight after they are made
    #and the projectiles only need to be created once as they can be launched many times
    if len(self.sightProjectiles) == 0:
        self.createProjectiles()

    #This for loop goes through each projectile in the self.projectiles list
    for projectile in self.sightProjectiles:
        #This if statement checks whether the projectile has yet to be launched for the first time or if it has collided with the wall and finished
        if projectile.getCollided() == True or projectile.getLaunched() == False:
            #If either of the above is true then it sets the collided attribute of the projectile as false
            projectile.setCollided(False)
            #and then re-launches the projectile again to check, a list of objects it has collided with are returned
            collided = projectile.launchSightProjectile(screen, map, self.getHitbox().center)
            #This list is then appended to the self.collided list
            self.collided.append(collided)

    #This nested for loop goes through each object that all the projectiles have collided with
    for projectile in self.collided:
        for object in projectile:
            #This sets the visibility of the object as true so that the player can see it
            object.setVisible(True)
    #The self.collided list has to be reset each time otherwise the previous objects would also be set as visible to the player
    self.collided = []

```

This is the fov method. It takes the parameters map – this is the map object, and the screen – screen object. First a variable called allObjects is set as the result of using the getAllObjects method on the map object – this method returns a one dimensional list of all of the objects in the map. Then a for loop which loops through each object in this list is used to set the visibility of all of the objects to false by using the setVisible method. This means that the object will appear as black on the player screen. Then an if statement is used to check if the projectiles have been created yet – if the length of the list of the sight projectiles is 0 that means that the projectiles have not been created as there are no projectiles in the list. Therefore, if the list is empty then the createProjectiles method is run. Next, another for loop is used which goes through each projectile in the list of sight projectiles. Then an if statement is used and it checks if the projectile has not been launched for the first time or if the projectile has already collided with a wall using the respective methods. This if statement is there to ensure that the projectile launch rate is limited so that it does not cause the program to run slower. Then the collided attribute of the projectile is set to false using the relevant method, this is to reset the projectile if it has collided with a wall before, otherwise the while loop in the launchProjectile method would not work. After this the projectile is launched using the launchSightProjectile method which is passed the screen, the map, and the center of the hitbox of the player as the coordinates so that the projectiles originate from the player. The value returned from this method is a list of all of the objects which the projectile has collided with and is appended to the new collided attribute of the player which is also a list. Then a nested for loop is used which goes through each of the objects which the projectiles have collided with and then sets the visibility as True. This allows the player to see the objects. Finally the collided attribute is reset to an empty list otherwise the previously visible objects would still be visible.

```
def getCollided(self):
    return self.collided

def ready(self, map, screen):
    self.displayPlayer(screen.getScreen())
    self.fov(map, screen)
    self.checkCollision(map)
```

These are the last 2 methods which I have added so far for the player class. The getCollided method returns the collided attribute for the player which is a list. The ready method runs all of the methods needed for the player to be able to work, except for the checkKeys method as this needs to be run within a different part of the game loop in the main program

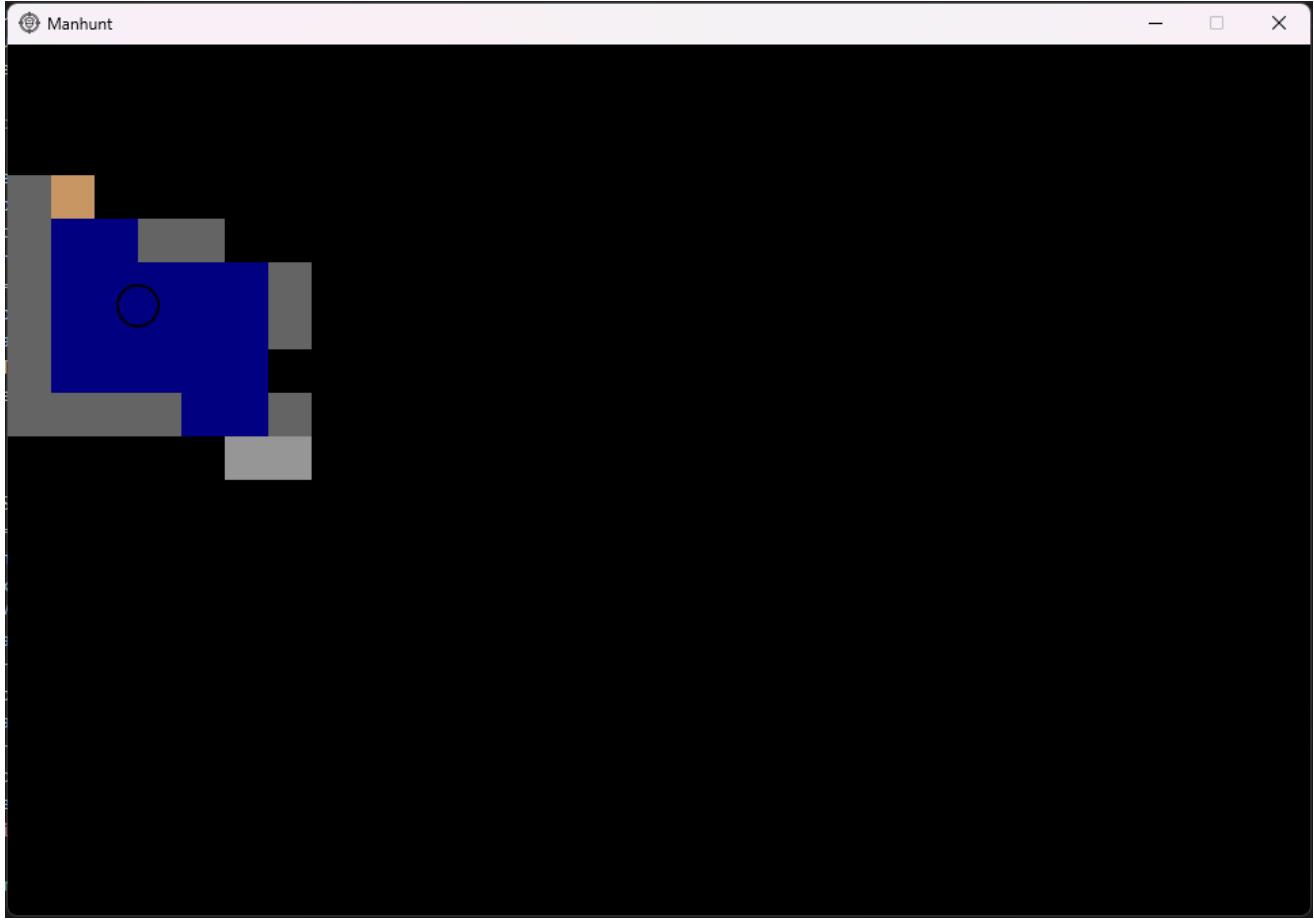
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



This is testing the field of view for the player, the expected output is that the player can see everything in the room and some parts out of the room but nothing beyond the walls, the top image is with the projectiles visible to check whether they are going in the correct place and the second one is what it looks like without the projectiles being displayed

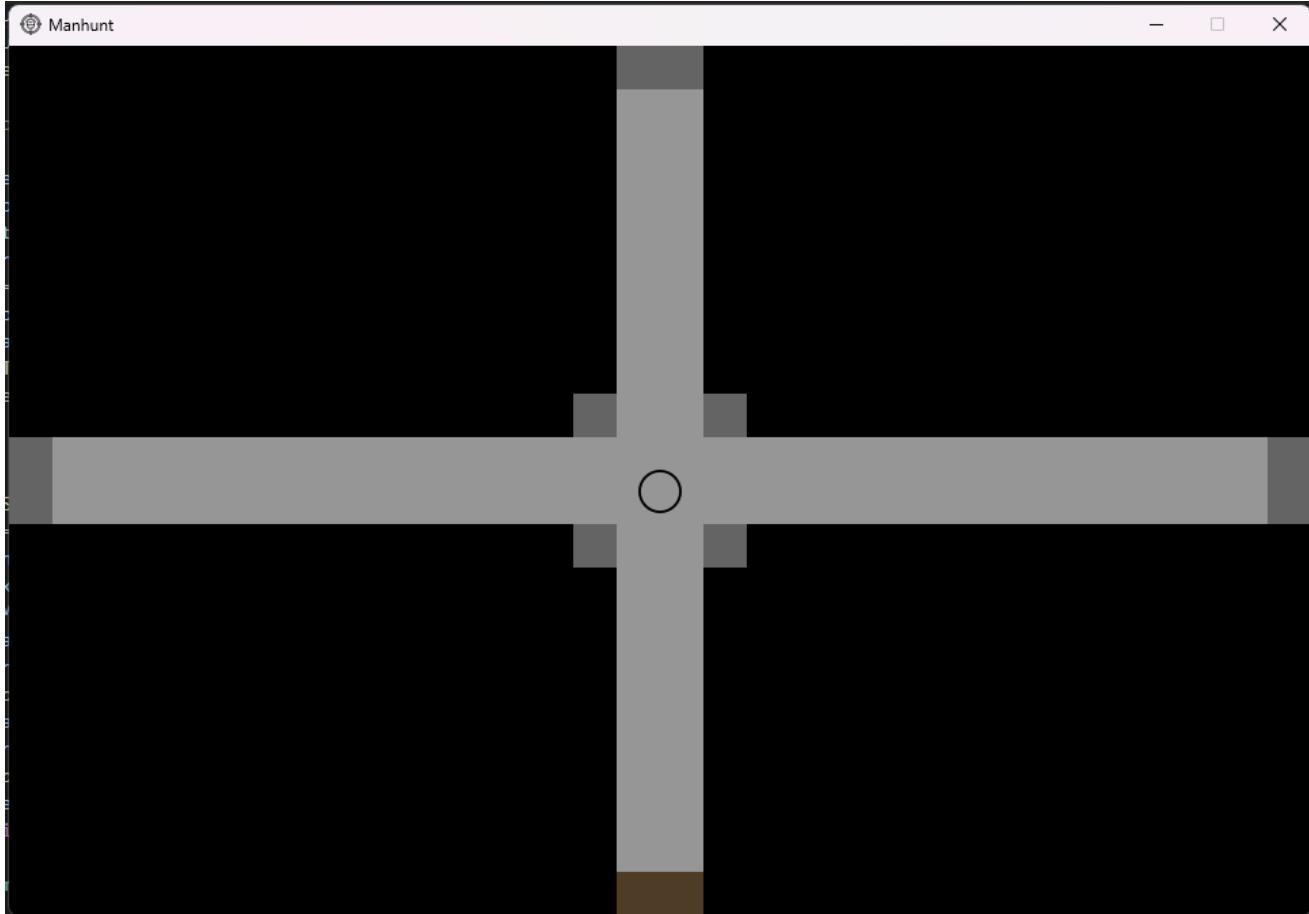
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

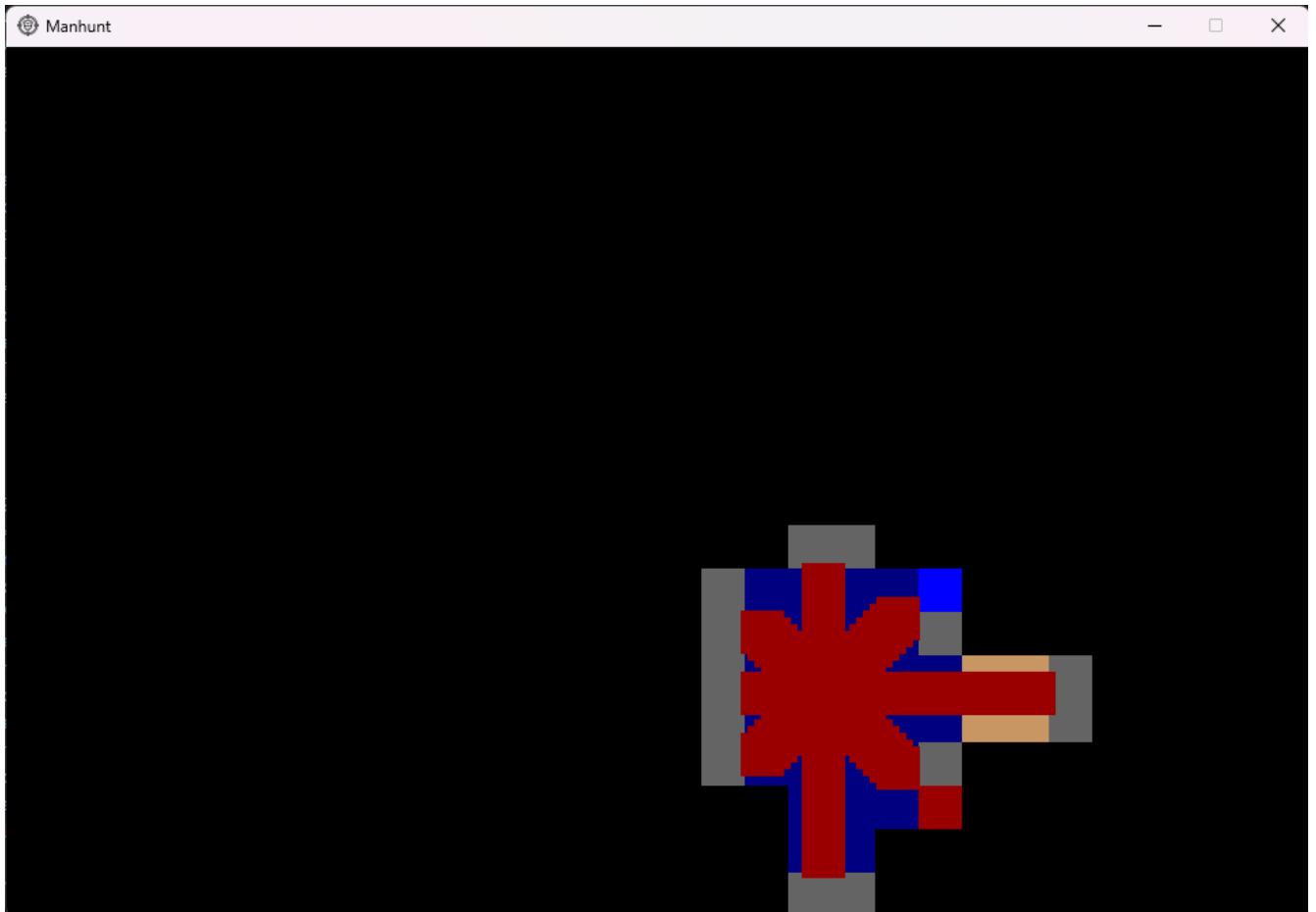
Project title: Manhunt

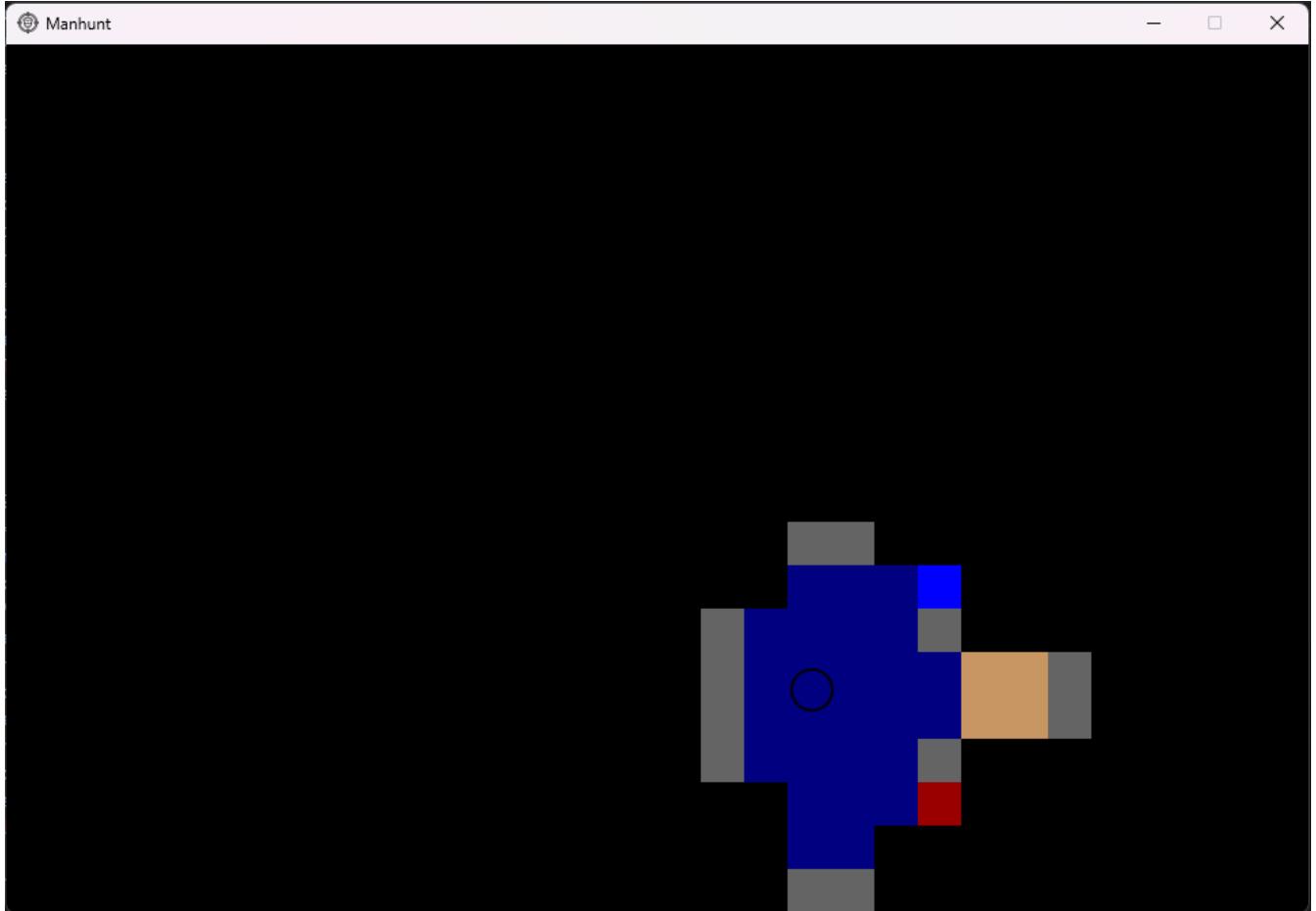


This is testing whether the projectiles can reach the end of the hallway and only show that when standing in the middle of the map, it is also to check whether the game will be slow if I go into the middle since there will be a lot more projectiles since there is a larger amount of space where the projectiles have to travel. Previously when I moved to the middle, the game would slow down by a large amount

Name: Muqtasid Zayyan Dar

Project title: Manhunt





In this image I am testing a different part of the map to see whether it is possible to see the different colours of the hiding space (blue) and the lever (red). This test shows that the FOV for the player works but some parts are dark although, realistically, the player should also be able to see those corners of the room. If I add more projectiles to the game, it may slow it down, therefore, I am going to use this as the player's FOV for now.

Review:

In this iteration of the program I used the sight projectile class which I previously created in order to create the field of view for the player, although I believed the solution would be quite efficient, there are still lag spikes when the player walks into large areas which make the program run slower. However, I think that this is not very noticeable and does not cause any major issues. However, just before I create the hunter class, I need to derive a sprite class from the player in order to make it more efficient to code since the hunter can inherit most of the methods from the sprite class.

Specs completed/removed:

- P21, O11 – These have been completed as the player now has 360 vision but cannot see through the walls

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, P8, P9, P20, Ae4, O9

Specs started:

- None

Version 5.7:

In this sub version I am going to derive a sprite class from the player class which will then be used as a super class for both the hunter and the player. I am also going to finish off the sound part of the player and fix the map coordinates for the player.

```
class Sprite():
    def __init__(self, x, y, img, scale, speed, sprintMultiplier):
        self.x = x
        self.y = y
        self.mapX = round(self.x/32)
        self.mapY = round(self.y/32)
        #Image attributes
        self.img = pygame.image.load(img).convert_alpha()
        width = self.img.get_width()
        height = self.img.get_height()
        self.transformedImg = pygame.transform.scale(self.img, (int(width * scale), int(height * scale)))
        self.width = self.transformedImg.get_width()
        self.height = self.transformedImg.get_height()
        #Hitbox of the player by creating a rectangle around the player img
        self.hitbox = self.transformedImg.get_rect()
        #Speed in any direction
        self.speed = speed
        #This is the sprinting multiplier which increases the speed of the player by that much
        self.sprintMultiplier = sprintMultiplier
        self.forward = False
        self.backward = False
        self.left = False
        self.right = False
        self.sprinting = False
```

These are all of the attributes which I have included in the sprite superclass, I have added an attribute for sprinting as I need to be able to detect when the sprite is sprinting so that I can decide when to apply the sprint multiplier to the speed. Most of the other attributes are copied from the player class

```
#This method moves the player up by a specific number of pixels in the negative Y direction
def moveForward(self):
    self.forward = True
    if self.sprinting == True:
        self.y -= self.speed * self.sprintMultiplier
    else:
        self.y -= self.speed

#This method moves the player up by a specific number of pixels in the positive Y direction
def moveBackward(self):
    self.backward = True
    if self.sprinting == True:
        self.y += self.speed * self.sprintMultiplier
    else:
        self.y += self.speed

#This method moves the player up by a specific number of pixels in the negative X direction
def moveLeft(self):
    self.left = True
    if self.sprinting == True:
        self.x -= self.speed * self.sprintMultiplier
    else:
        self.x -= self.speed

#This method moves the player up by a specific number of pixels in the positive X direction
def moveRight(self):
    self.right = True
    if self.sprinting == True:
        self.x += self.speed * self.sprintMultiplier
    else:
        self.x += self.speed
```

These are the sprite version of the movement methods, key presses do not need to be checked because only the player moves based on certain key presses, however, the hunter will move based on a different algorithm so there is no need to check for key presses in these methods.

```
def setSprinting(self, value):
    if value == True or value == False:
        self.sprinting = value
    else:
        print('NOT RIGHT DATA TYPE FOR SPRINTING')

def getSprinting(self):
    return self.sprinting
```

These are the get and set methods for the sprinting attribute. For the set method, I added validation in order to ensure that the value passed into the sprinting method was the correct one

```

def getForward(self):
    return self.forward

def getBackward(self):
    return self.backward

def getLeft(self):
    return self.left

def getRight(self):
    return self.right

```

These are the get methods for each of the directional attributes, they all return their respective directional attribute

```

def setCoords(self):
    self.hitbox.center = (self.x, self.y)

```

This is the setCoords method, it was copied directly from the player class as it does not differ between the player and the hunter

```

#This returns the coordinates of the player divided by 32 as the size of the map is a factor of 32 compared to the size of the screen
def getMapCoords(self):
    x = round(self.hitbox.center[0]/32)
    y = round(self.hitbox.center[1]/32)
    return (x, y)

```

This is the new working version of the getMapCoords method, previously I was trying to use the x and y in a calculation similar to try and return a value, however, it returned the same value no matter where the player was on the map. However, this new way uses the coordinates of the center of the player in order to calculate the map coordinates. Each value is divided by 32 as I have created the ratio of the objects in the map to the pixels on the screen as 32:1.

```
#This returns the center of the hitbox which is in the same place as the player
def getCoords(self):
    return self.hitbox.center

#This is the function which checks for collisions of the player with walls and then causes the player to bounce off of it
def checkCollision(self, map):
    #This function returns all of the walls for the map in a list
    walls = map.getWalls()
    #This is how many pixels the player will bounce off the wall when collision occurs
    collisionBounce = 5
    #This loop goes through each wall in the list above
    for wall in walls:
        #Gets the rectangle of the wall
        rect = wall.getRect()
        'print(str(player.colliderect(rect)))'
        #Checks if the rectangle of the player has collided with that rectangle (for the wall)
        if self.hitbox.colliderect(rect):
            '#print('COLLIDING')
            #Each of these if statements check which direction the player is currently moving
            #and based on this decision, the direction the player bounces back is determined
            if self.backward == True:
                '#print('BOTTOM')
                self.y -= collisionBounce

            if self.forward == True:
                '#print('TOP')
                self.y += collisionBounce

            if self.right == True:
                '# print('RIGHT')
                self.x -= collisionBounce

            if self.left == True:
                '#print('LEFT')
                self.x += collisionBounce

    def getHitbox(self):
        return self.hitbox

    def displaySprite(self, screen):
        screen.blit(self.transformedImg, self.hitbox.topleft)
```

These methods are all copied from the player class and have already been explained.

```
def checkSound(self, map):
    coords = self.getMapCoords()
    floor = map.getObject(coords)
    if isinstance(floor, objects.Floor):
        sound = floor.getSoundLevel()
        self.setSound(sound)
    else:
        print('NOT FLOOR')

#This sets the sound that the player is making based on the coordinates of the player on the map
def setSound(self, value):
    if value <= 0 or value >= 1:
        print('INCORRECT VALUE')
        return
    self.sound = value

#This returns the sound level for the player
def getSound(self):
    return self.sound
```

These are the sound methods. The top method gets the coordinates of the player on the map and then

uses these coordinates in another method on the map object in order to get the object in that coordinate on the map – this is given the identifier floor. Then 'floor' is validated in order to make sure that it is a floor type object – this is because earlier while testing, I noticed an error where, due to the nature of how I coded it, the player sometimes glitches into the wall slightly and causes an error. After validating the floor then the sound level of the floor is set as the sound for the player's sound attribute. The set sound method validates the value passed into the method as the value of the sound can only be less than 1 but greater than 0 for the calculations then it sets this value as the sound attribute. Finally there is the get method for the sound attribute which just returns the sound attribute

Review:

Although initially I was thinking of using the sprite class from the pygame library, I thought it would be more efficient to code my own version of the sprite library with only the necessary methods so that I would not have to learn what all of the pygame methods do and it would be easier for me to figure out where errors may be in my own methods rather than in the pygame methods. I also learnt that it is better to create parent classes before as it was much more hassle deriving the sprite class from the player class since I had to generalise some methods in the sprite class and then change them in the player class.

Specs completed/removed:

- None

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, P8, P9, P20, Ae4, O9

Specs started:

- None

Version 6:

In this version I am going to code the hunter class which will also be inheriting the sprite class which I will be deriving from the player class.

Version 6.1:

In this sub version I am going to create the detection system to count the number of walls between the player and the hunter and also the calculations for the chance for the hunter to hear the player.

```
class Hunter(sprite.Sprite):
    def __init__(self, x, y, img, scale, speed, sprintMultiplier):
        super().__init__(x, y, img, scale, speed, sprintMultiplier)
        self.soundProj = None
```

This is the hunter class, it inherits all of the methods and attributes from the sprite class which I previously derived from the player class. It also has most of the attributes which the player class has as well. The only new attribute at the moment in the soundProj attribute, this attribute will be used to hold a single sound projectile object which will be used in order to check the number of walls between the player and the hunter.

```
#This method calculates the distance in each axis
def coordinateDistance(self, coords):
    hCoords = self.getMapCoords()
    print(coords)
    print(hCoords)
    startX = coords[0]
    endX = hCoords[0]
    startY = coords[1]
    endY = hCoords[1]
    xDist = startX - endX
    yDist = startY - endY
    return xDist, yDist
```

This is the coordinate distance method which takes the goal coordinates as a parameter which is a tuple.

It first gets the coordinates of the hunter on the map in the form of a tuple and gives it an identifier. The next 2 print statements are just for testing, however, the lines after that take each relevant part of the tuple and then gives it an identifier to make it easier to figure out what each value is. Then the distance from the hunter to the goal coordinates are calculated in each dimension by using the goal coordinate as the start point and the hunter coordinate as the end point. Finally both dimensional distances from the hunter to the player are returned

```
def createSoundProjectile(self):
    self.soundProj = physics.SoundProjectile(self.getHitbox().center, 0, 0, 32)
```

This is the createSoundProjectile method, it creates a sound projectile object and passes it the coordinates of the center of the hunter's hitbox, sets the x and y speed as 0 initially and then passes a length of 32.

```
def ready(self, screen, map, player):
    self.setCoords()
    self.displayHunter(screen)
    self.checkWalls(player, screen, map)
```

This is the ready method, it takes parameters: screen – a surface object, map – a map object and player – a player object. It runs various methods which the hunter needs in order to be able to function in the game. The first method updates the hunters coordinates. The second method displays the hunter on the screen and the final method checks the number of walls between the player and the hunter, although this method is here initially, it will be embedded in a different method later on which calculates the chance of the hunter hearing the player.

```
#This method checks the number of walls between the player and the hunter
def checkWalls(self, player, screen, map):
    #Checks if the sound projectile object has been created
    if self.soundProj == None:
        self.createSoundProjectile()
    #Sets the attribute for the sound projectile as a variable
    soundProj = self.soundProj
    #Gets the player coordinates on the map
    pCoords = player.getMapCoords()
    #Calculates the distance between the player and the hunter in each axial direction
    coordDist = self.coordinateDistance(pCoords)
    print(coordDist)
    xSpeed = 0
    ySpeed = 0
    #Sets the speed of the sound projectile based on which direction the player is in
    if coordDist[0] < 0:
        xSpeed = -5
    else:
        xSpeed = 5
    if coordDist[1] < 0:
        ySpeed = -5
    else:
        ySpeed = 5
    print(xSpeed)
    print(ySpeed)
    soundProj.setXSpeed(xSpeed)
    soundProj.setYSpeed(ySpeed)
    #Runs a method on the sound projectile which returns the number of walls the projectile has collided with
    walls = soundProj.launchSoundProjectile(screen, self.getHitbox().center, map, player)
    return walls
```

This is the first iteration of the checkWalls method. This method takes a player object, the screen surface to display objects and a map object as parameters. First the method checks if a sound projectile has been created, if not then a sound projectile is created using the createSoundProjectile method. Then the sound projectile attribute is given an identifier to make it easier to refer to later on. Then a method is used on the player object to get the map coordinates for the player in the form of a tuple – This is given the identifier pCoords. Then a method is used in order to calculate the distance from the hunter to the player in each dimension (x and y). After this, depending on whether the coordinate distance is positive or negative in each dimension, the speed of the projectile is set as being the same sign so that the projectile travels at a set speed in that direction. I thought that this would make the projectile travel towards the player, however, I was wrong. Continuing with what the rest of the method does, the print statements are just for testing and then the next 2 lines after that set the x speed and y speed of the sound projectile. The next line uses a method to launch the sound projectile and is passed a surface object (to display the sound projectile for testing) the coordinates of the center of the hunter, the map object and the player object. The result of this is given the identifier walls as it returns an integer which is the number of walls which the player has collided with. When I tested this code, the game kept crashing as it was infinitely looping, the error was due to my logic. I completely overlooked the fact that by using the exact same value for the x and y speed in either direction, the projectile would not follow the player but instead just travel at a 45 degree angle from the hunter towards the part of the map the player was in – this meant that the sound projectile never hit the player which caused the game to infinitely loop since the projectile must hit the player in order for the wall checking loop to stop. In order to prevent this I need to validate the sound projectiles to make sure that they stop also when they collide with the borders on the screen. The next few methods are a new solution which I thought of that would work.

```
#This method returns what values the xSpeed and ySpeed of the sound projectile should be
def directionCheck(self, xDist, yDist):
    xSign = 0
    ySign = 0
    if xDist < 0:
        xSign = -1
    elif xDist > 0:
        xSign = 1
    if yDist < 0:
        ySign = -1
    elif yDist > 0:
        ySign = 1
    return xSign, ySign
```

This is the directionCheck method which takes the x and y distances of the player to the hunter and then decides what direction the projectile should be going in (what the sign for the speed should be)

```
#This method takes in the absolute values for the x and y distances and then returns
#the angle at which the projectile needs to be launched
def angleCalcR(self, xDist, yDist):
    o = abs(yDist)
    a = abs(xDist)
    value = o/a
    angle = math.atan(value)
    return angle
```

This method calculates the angle at which the projectile needs to be shot at using the absolute values of the x and y distances from the player to the hunter – This was just to avoid any unnecessary problems with the direction of the velocity of the projectile as the y axis works in a different manner in this situation compared to convention. Then trigonometry is used in order to find the angle at which the projectile needs to be launched at.

```
#This method takes the angle at which the projectile needs to be launched and
#uses trigonometry in order to calculate what the x and y component need to be
#in order for the projectile to move at a certain speed in that direction
def speedCalcR(self, angle):
    xSpeed = 0
    ySpeed = 0
    projSpeed = self.soundProjSpeed
    cos = math.cos(angle)
    sin = math.sin(angle)
    xSpeed = projSpeed * cos
    ySpeed = projSpeed * sin
    return xSpeed, ySpeed
```

This method takes the angle in radians which has been calculated as a parameter. It then initializes 2 variables for x speed and y speed as 0 and a third variable in order to make it easier to refer to the sound projectile speed lower down in the method. Next math functions are used with the angle and

then multiplied by the sound projectile speed in order to calculate the speed for each of dimensional components for the projectile

```
import hunter
import pygame
import random

pygame.init()

hunterOne = hunter.Hunter(0, 0, 'RedCircle.png', 1, 10, 13)

proj = False
goalDist = (random.randint(-20, 20), random.randint(-20, 20))

def checkWalls(hunter, goalDist):
    if proj == False:
        print('Created proj')
    dist = hunter.coordinateDistance(goalDist)
    print('dist', dist)
    xDist = dist[0]
    yDist = dist[1]
    direction = hunter.directionCheck(xDist, yDist)
    print('direction', direction)
    angle = hunter.angleCalcR(xDist, yDist)
    print('angle', angle)
    speeds = hunter.speedCalcR(angle)
    xSpeed = speeds[0] * direction[0]
    ySpeed = speeds[1] * direction[1]
    print(xSpeed, ySpeed)

checkWalls(hunterOne, goalDist)
```

This was the code I used in order to test the new methods above in a new procedure outside of the class called checkWalls. This was so that I could have a rough idea of how to code it within the class. First I imported the necessary files and then created a hunter object and gave it the identifier of hunterOne. Next I set a variable called proj to false and created a tuple which has 2 randomised integers which will be used as the goal distance which would be in the method. The procedure takes the hunter object and the tuple as parameters. Then it first uses an if statement in order to check if a projectile is created and if it is false (which it always is), then it will print something to show it has created a projectile. Then the method coordinateDistance is used in order to calculate the distance from the hunter to the goal

coordinates – This is given the identifier dist as it returns a tuple. Then each element of that tuple is given a different identifier as they are for the distance in the different dimensions. Next the direction of the velocity for the projectile is checked using a method which takes the variables created above and returns a different tuple which will be used later on in order to change the direction of the projectile. Next the angle of the projectile is calculated using the method angleCalcR which is passed the same 2 variables and returns the angle at which the projectile needs to be launched in radians. Next the speedCalcR method is used which takes the angle which the projectile needs to be launched in radians and then returns the x and y speed that the projectile needs to be going to travel at a specific speed at that angle. Finally, the results of the direction check are multiplied to each of the x and y speed in order to make sure that the components of the projectile are in the right direction. Then this result is printed out. The last line at the bottom just invokes the function above.

Created proj	Created proj
(-11, -17)	(8, -4)
(0, 0)	(0, 0)
dist (-11, -17)	dist (8, -4)
direction (-1, -1)	direction (1, -1)
angle 0.9964914966201949	angle 0.4636476090008061
-2.7162563907863713	-4.197850785760756
4.47213595499958	-2.23606797749979
Created proj	Created proj
(-2, 13)	(2, 4)
(0, 0)	(0, 0)
dist (-2, 13)	dist (2, 4)
direction (-1, 1)	direction (1, 1)
angle 1.4181469983996315	angle 1.1071487177940904
-0.7602859212697051	4.941858488253086
2.2360679774997902	4.47213595499958

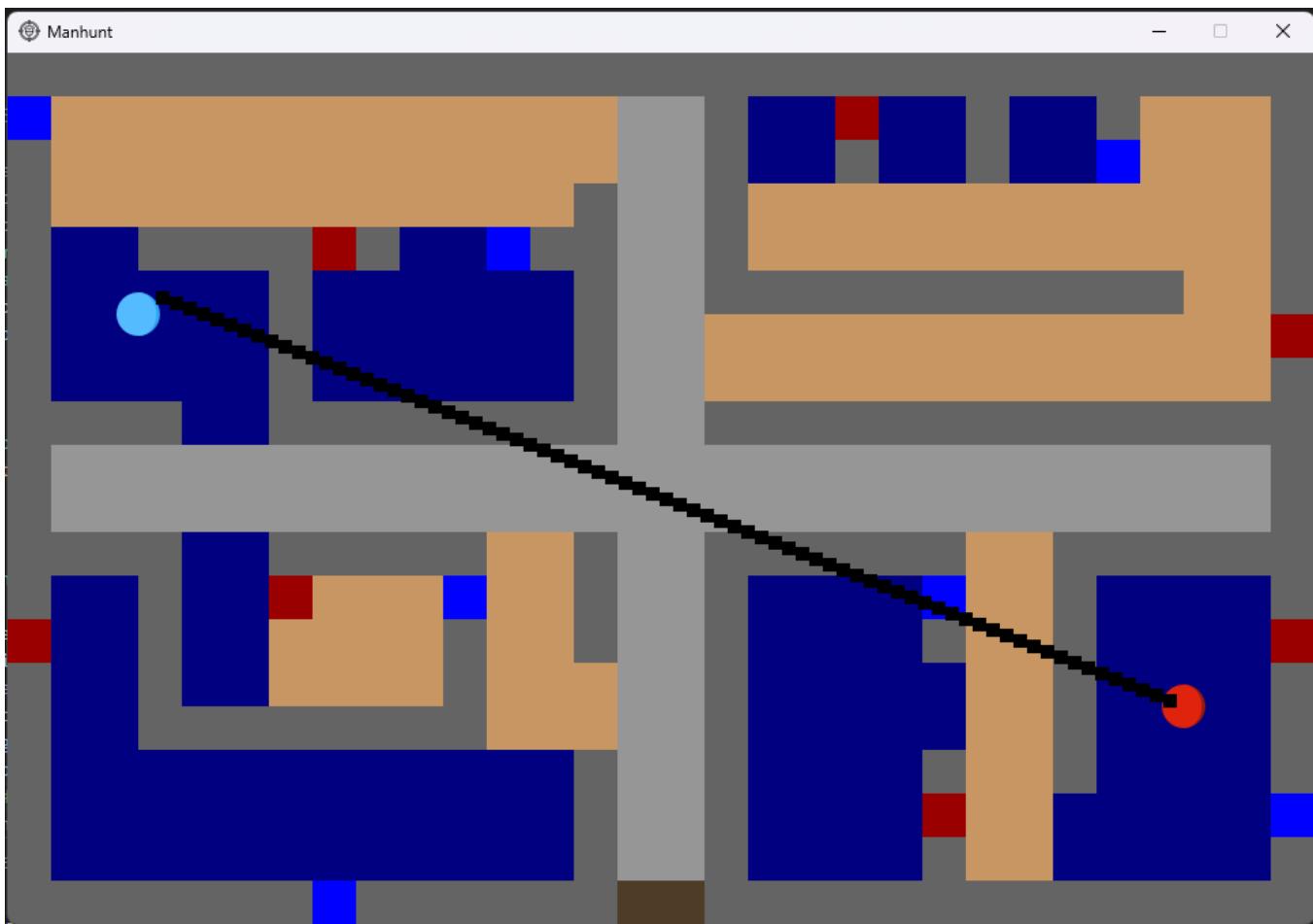
This is the result of running the code a few times. The first value is the goal coordinates and the second is the hunter coordinates. After that is the distance from the hunter to the goal which is the same as the goal coordinates is printed out, this is because the hunters start coordinates are always 0. Next the direction of the velocity of the projectile is printed out. After that the angle in radians is printed out. Finally, the x and y velocity of the projectile is printed out. In all of the tests, the outcome was exactly what I expected after coding it.

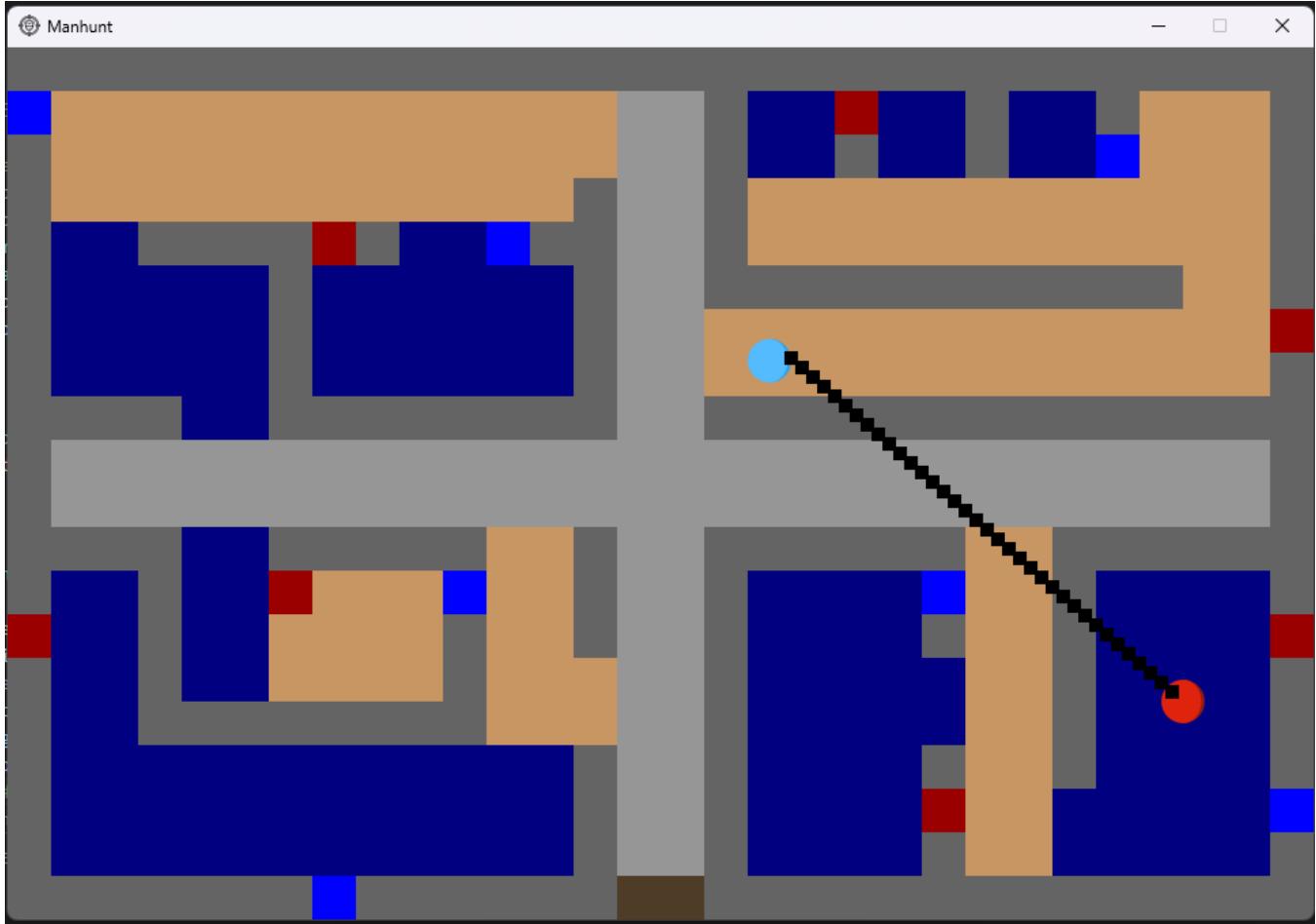
```
#This method checks the number of walls between the player and the hunter
def checkWalls(self, player, screen, map):
    #Checks if the sound projectile object has been created
    if self.soundProj == None:
        self.createSoundProjectile()
    if self.soundProj.getLaunched() == False or self.soundProj.getCollided() == True:
        self.soundProj.setCollided(False)
    #Sets the attribute for the sound projectile as a variable
    soundProj = self.soundProj
    #Gets the player coordinates on the map
    pCoords = player.getCoords()
    #Calculates the distance between the player and the hunter in each axial direction
    coordDist = self.coordinateDistance(pCoords)
    xDist = coordDist[0]
    yDist = coordDist[1]
    direction = self.checkDirection(xDist, yDist)
    print('direction', direction)
    print('doing')
    angle = self.angleCalcR(xDist, yDist)
    print('angle', angle)
    speeds = self.speedCalcR(angle)
    xSpeed = speeds[0] * direction[0]
    ySpeed = speeds[1] * direction[1]
    print(xSpeed, ySpeed)
    soundProj.setXSpeed(xSpeed)
    soundProj.setYSpeed(ySpeed)
    walls = soundProj.launchSoundProjectile(screen, self.getHitbox().center, map, player)
    return walls
else:
    return
```

This is the new checkWalls method which I created after using the new code I created, any print statements in this method are purely for testing in order to make sure that each part works correctly. This method takes the player object, surface object and map object as parameters. Then first it checks if a sound projectile has been created and if it hasn't been created yet then the method is called which creates a projectile. Next an if statement is used in order to check whether the sound projectile has launched or has collided with the player or border walls. If either of these are true then the projectile's collided attribute is reset to false so that it can be used, then the sound projectile attribute is given an identifier in order to make it easier to refer to later on. Then after that, a method is used on the player coordinates in order to get the screen coordinates of the player. After this, the rest of the functions are copied from the tested one above, then the next difference is after setting the xSpeed and ySpeed variables where these variables are then passed into a method for the sound projectile in order to set this as the x and y speed for the projectile. Finally, the launchSoundProjectile method is run and is passed the screen, the coordinates of the center of the hunter, the map object and the player object. The outcome of this is then given the identifier walls which is returned.

Name: Muqtasid Zayyan Dar

Project title: Manhunt





This is how I tested the sound projectile, in the code I used a method in order to make the projectiles visible and then I moved across the map. When I did this I expected the projectiles to follow the player directly from the hunter to the player. However, when I did this then the projectiles mostly followed the player which is good but sometimes, the change in the angle is too small to reflect a change in the projectiles direction so the projectiles sometimes only get a rough estimate rather than exactly from the hunter to the player – this is a limitation of my program since the speed of the projectile can only be so high so the precision of the x and y speeds are also limited, therefore, this is a good enough version of the projectile for my game. I also tested different speeds and sizes for the projectile, I noticed that the higher the speed for the projectile, the more accurate it was when I moved the player around and the slower it was, the less accurate, however, due to the limitation of the projectile only being able to be a certain maximum speed so that it doesn't skip over any walls the highest speed I could use would be 16.

```

File "c:\Users\Muqtasid\OneDrive\Documents\GitHub\Manhunt\Manhunt - Game Code.py", line 175, in gameScreen
    hunterWin = hunterOne.ready(game.getScreen(), map, playerOne)
File "c:\Users\Muqtasid\OneDrive\Documents\GitHub\Manhunt\hunter.py", line 129, in ready
    self.checkWalls(player, screen, map)
File "c:\Users\Muqtasid\OneDrive\Documents\GitHub\Manhunt\hunter.py", line 80, in checkWalls
    angle = self.angleCalcR(xDist, yDist)
File "c:\Users\Muqtasid\OneDrive\Documents\GitHub\Manhunt\hunter.py", line 44, in angleCalcR
    value = o/a
ZeroDivisionError: division by zero
  
```

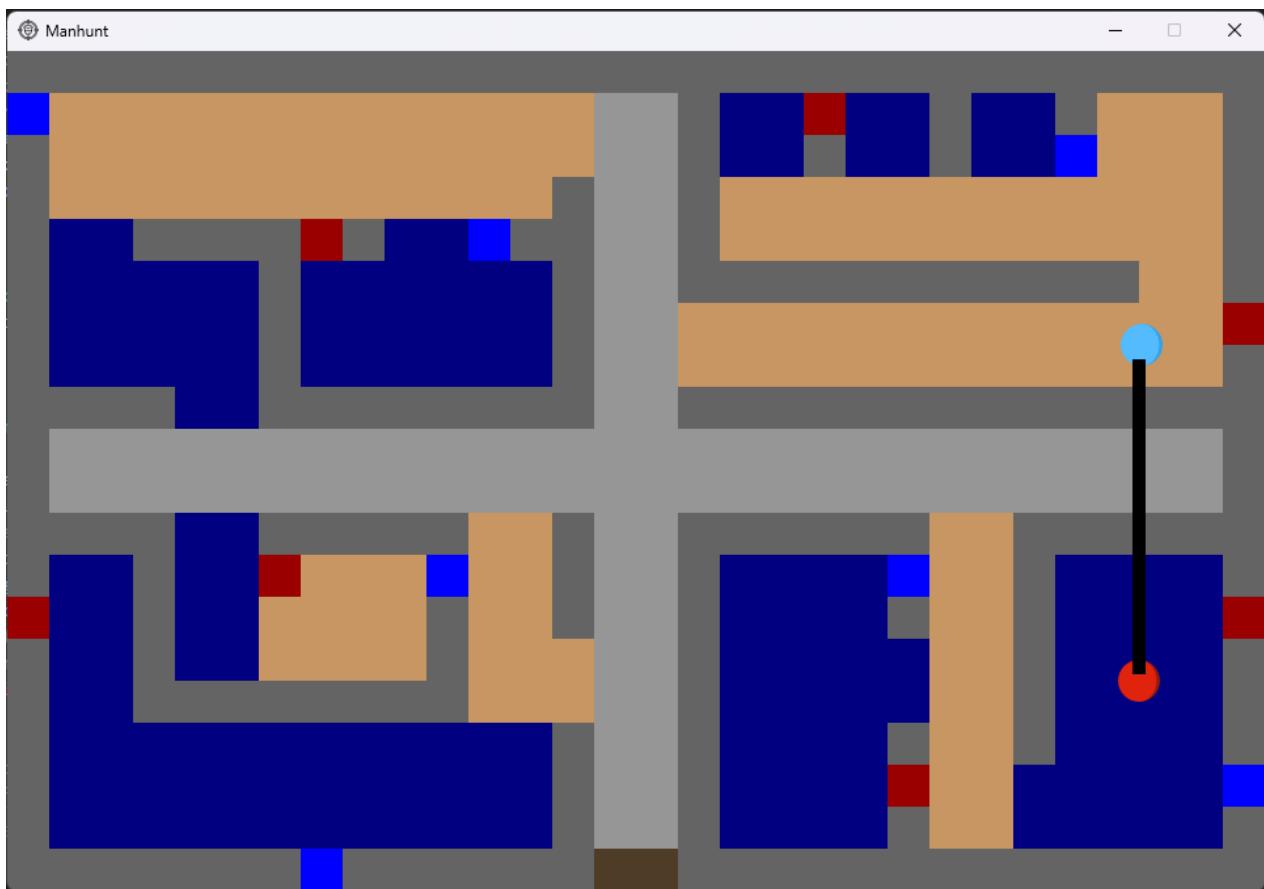
One problem I came across when moving the player around was when the player was directly in line with the hunter as the value for one of the distances from the player to the hunter would be 0 so I would by attempting to divide by 0 which is impossible and needs to be fixed. I will most likely do this by adding some lines which check if either the x or y distances are 0 and then carry out something based on that.

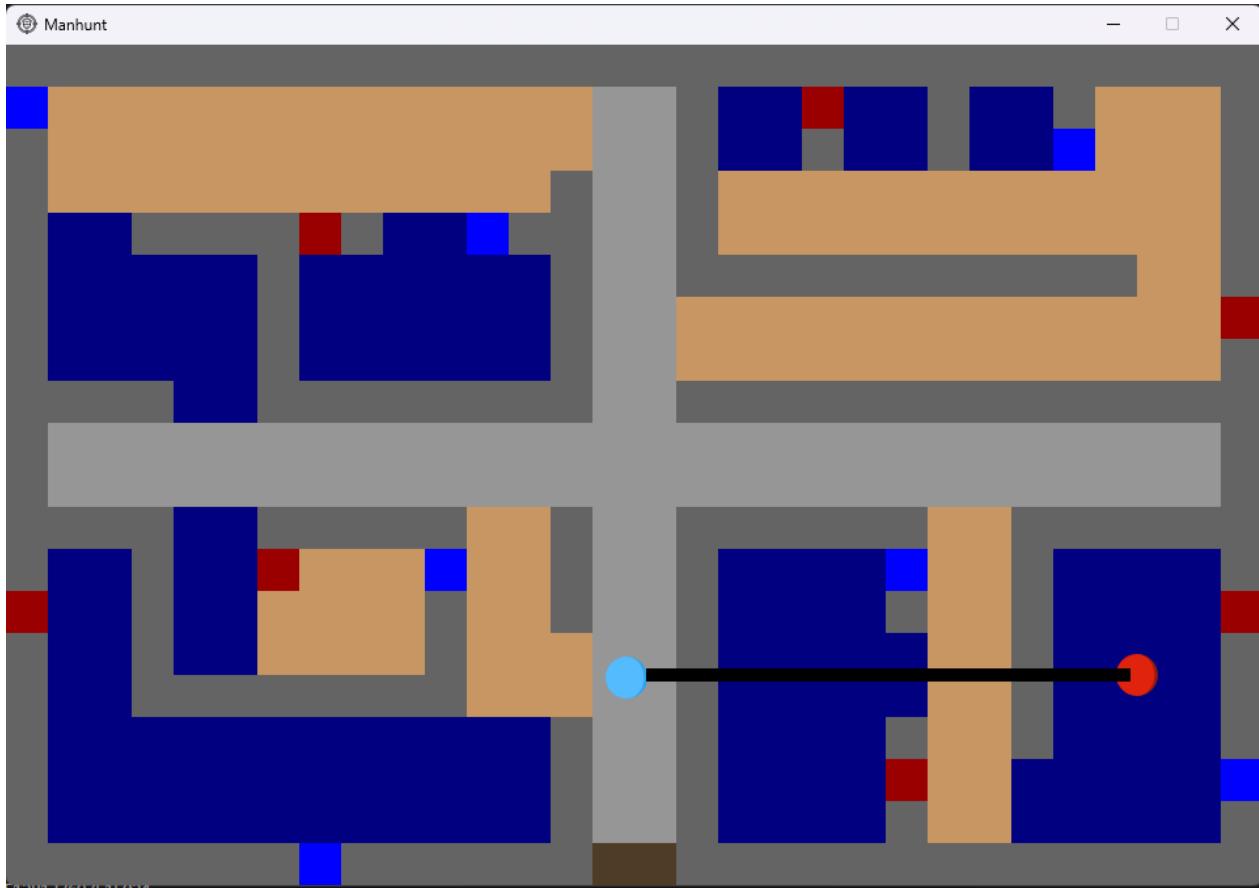
```
if xDist != 0 and yDist != 0:  
    print('direction', direction)  
    print('doing')  
    angle = self.angleCalcR(xDist, yDist)  
    print('angle', angle)  
    speeds = self.speedCalcR(angle)  
    xSpeed = speeds[0] * direction[0]  
    ySpeed = speeds[1] * direction[1]  
elif xDist == 0:  
    xSpeed = 0  
    ySpeed = self.soundProjSpeed * direction[1]  
elif yDist == 0:  
    ySpeed = 0  
    xSpeed = self.soundProjSpeed * direction[0]
```

This is the edit which I made to the code, It checks to make sure the neither the x or y distance from the hunter to the player is 0 and if it is then based on the direction the direction method calculates, either the x or y speed is 0 and the other component of speed is the maximum

Name: Muqtasid Zayyan Dar

Project title: Manhunt





These are the tests since I made the change which was exactly what I expected when the player is in line with the hunter. However, when I printed out the number of walls which the function counted as being in-between the player and the hunter, it was not what I expected and the number was a lot higher than it should have been.

```
def wallCheck(self, walls):
    for wall in walls:
        if self.collideCheck(wall) == True and self.alreadyCollided(wall) == False:
            self.wallNum += 1
            self.collidedWalls.append(wall)

def alreadyCollided(self, check):
    for wall in self.collidedWalls:
        if check == wall:
            return True
    return False
```

In order to fix the problems where the wall were being counted multiple times, I decided to include a new attribute in the soundProjectile class which is a list of the walls which the projectile has collided with, now once the projectile has collided with a wall the alreadyCollided method will check whether the projectile has already collided with the wall using a simple linear search to search the collidedWalls

list and then if it has not already collided with that wall then the number of walls is incremented by 1 and the wall is appended to that list.

```
def sound(self, player, screen, map):
    maxSound = 100
    soundLevel = player.getSound()
    if isinstance(soundLevel, float) and player.getMovement():
        maxSound = soundLevel * maxSound
        wallNum = self.checkWalls(player, screen, map)
        maxSound -= wallNum
        if maxSound <= 50:
            return False
        elif maxSound <= 75:
            chance = random.randint(51, 75)
            if chance == maxSound:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

This is the final sound method which takes the player object, a surface object and a map object as parameters. First a variable is set called maxSound, this variable will be used to determine how well the hunter is able to hear the player. Next, a method is used on the player in order to get the sound multiplier from the floor which the player is on – this is given the identifier soundLevel. Then the variable is checked to make sure that it is a float value and another method is also used on the player to check if the player is moving because the hunter should only be able to hear the player if the player is moving. Then, the maxSound is multiplied by the factor from the soundLevel variable. After this, the checkWalls method is used to count the number of walls between the player and the hunter which is returned in the form of an integer. Then the number of walls between the hunter and the player is subtracted from the maxSound value. Then depending on the value of maxSound, a boolean value is returned which will determine whether the hunter has heard the player or not.

Review:

In this iteration of the program, I learnt how I can use maths which I have learnt and implement it into the game in order to get the outcome I need. As the hearing for the hunter is now done, the next part which I believe needs to be done is the sight for the hunter.

Specs completed/removed:

- P10 – I have decided to change this criterion as the hunter's sound detection will not be determined by overlap of circles but rather by an arbitrary calculation which will also use probability. Therefore, this is now completed as it has been implemented in the sound method

- Ae4 – I have given the hunter a sprite which looks like the player's but is a red colour instead of blue therefore, this is completed
- P9 – I have decided to change this criterion as I am now using projectiles in order to determine the number of walls between the player and the hunter and the rest is based on calculations, therefore, this is also now complete with the addition of the checkWalls method

Specs ongoing:

- P7, P12, P18, P24, O6, O10, O12, P19, P8, P20, O9

Specs started:

- P4, P5

Version 6.2:

In this sub version I am going to be coding the field of view for the hunter which should only exist in the direction which the hunter is moving in.

```
class HunterSightProjectile(Projectile):
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__(coords, xSpeed, ySpeed, length)
        self.playerCollision = False

    #This method checks if the projectile has collided with any walls or the player
    #and if it has then it returns the collided attribute as True and
    #if the projectile has collided with the player then the playerCollision attribute is
    #also set as True
    def checkCollisions(self, allWalls):
        for wall in allWalls:
            if isinstance(wall, player.Player):
                rect = wall.getHitbox()
            else:
                rect = wall.getRect()

            if self.collideCheck(rect):
                if isinstance(wall, player.Player):
                    self.collided = True
                    self.playerCollision = True
                else:
                    self.collided = True
```

First I created a new class for the sight projectile for the hunter, it inherits the projectile class and takes the parameters coords – tuple with 2 integers for coordinate which projectile originates from, xSpeed and ySpeed – these 2 parameters are used to control the x and y speed for the projectile but will not be set via the parameter and length – this is the length of each side of the rect for the projectile. It has one

additional attribute called playerCollision which will be used to monitor whether the projectile has collided with the player. The first method in this class is the checkCollisions method which takes the parameter allWalls which contains all of the walls in the map and the player object. Then a loop goes through each element within that list, for each element, the rect of that object is taken and then another method is used to check if the projectile has collided with that object. If it has then it will check whether that object is of type player – if it is then the collided attribute is set to True as it has collided with something that should stop the projectile and the playerCollision attribute is also set to True as the projectile has collided with a player object. If it has only collided with a wall then only the collided attribute is set to True so that the projectile stops and doesn't keep going through the wall.

```
#This method is very similar to the original sight projectile method
def launchSightProjectile(self, screen, coords, allWalls):
    self.rect.center = coords
    self.launched = True
    self.playerCollision = False
    while self.collided == False:
        self.moveProjectile(screen)
        self.displayProjectile(screen)
        self.checkCollisions(allWalls)
    return self.playerCollision

#This metho just returns the playerCollision attribute
def getPlayerCollision(self):
    return self.playerCollision
```

These are the other 2 methods within this class. The first method takes a screen object, coords as a tuple, and the allWalls list as parameters. It first sets the center of the rect for the projectile as the coordinates passed into the method. Then it sets the launched attribute as True as the projectile will have been launched. The playerCollision attribute is set to false so that it can be reset each time the method is called. Then a loop is in effect which runs while the collided attribute is false which means as long as the projectile has not collided with a wall or player object. The move projectile method within the loop moves the projectile across the screen and the displayProjectile method displays the projectile on the screen – this is only needed for testing and will be removed later on. Next is the checkCollisions method which was talked about above. At the end the playerCollision attribute is returned so that it can be used to check if the projectile has collided with the player.

```
self.sightProj = []
self.seen = False
```

These are the new attributes which I have added for the hunter's FOV, the sightProj attribute is a list which will hold all of the sight projectiles which the hunter will use in order to be able to see. The seen attribute will be used when the hunter has seen the player and will be set to True when that happens

```
def setSeen(self, value):
    if value != True and value != False:
        print('VALUE FOR SEEN IS NOT BOOLEAN')
    self.seen = value

def getSeen(self):
    return self.seen
```

These are the get and set methods for the seen attribute, the set method takes a value which must be a boolean value and then sets that as the value for the seen attribute and the get method returns the seen attribute.

```
def createSightProjectile(self):
    for x in range(0, 3):
        proj = physics.HunterSightProjectile(self.getHitbox().center, 0, 0, 32)
        self.sightProj.append(proj)
```

This is the createSightProjectile method which does not take any parameters and creates 3 HunterSightProjectile objects and appends these to the sightProj list

```

def setProjDirection(self, projSpeed):
    if self.getBackward():
        #LEFT
        self.sightProj[0].setXSpeed(projSpeed * -1)
        self.sightProj[0].setYSpeed(projSpeed)
        #MIDDLE
        self.sightProj[1].setXSpeed(0)
        self.sightProj[1].setYSpeed(projSpeed)
        #RIGHT
        self.sightProj[2].setXSpeed(projSpeed)
        self.sightProj[2].setYSpeed(projSpeed)

    elif self.getRight():
        #TOP
        self.sightProj[0].setXSpeed(projSpeed)
        self.sightProj[0].setYSpeed(projSpeed * -1)
        #MIDDLE
        self.sightProj[1].setXSpeed(projSpeed)
        self.sightProj[1].setYSpeed(0)
        #BOTTOM
        self.sightProj[2].setXSpeed(projSpeed)
        self.sightProj[2].setYSpeed(projSpeed)

    elif self.getLeft():
        #TOP
        self.sightProj[0].setXSpeed(projSpeed * -1)
        self.sightProj[0].setYSpeed(projSpeed * -1)
        #MIDDLE
        self.sightProj[1].setXSpeed(projSpeed * -1)
        self.sightProj[1].setYSpeed(0)
        #BOTTOM
        self.sightProj[2].setXSpeed(projSpeed * -1)
        self.sightProj[2].setYSpeed(projSpeed)

    else:
        #LEFT
        self.sightProj[0].setXSpeed(projSpeed * -1)
        self.sightProj[0].setYSpeed(projSpeed * -1)
        #MIDDLE
        self.sightProj[1].setXSpeed(0)
        self.sightProj[1].setYSpeed(projSpeed * -1)
        #RIGHT
        self.sightProj[2].setXSpeed(projSpeed)
        self.sightProj[2].setYSpeed(projSpeed * -1)

```

This is the setProjDirection method and based on the direction in which the hunter is moving, the x and y speed are changed so that the hunter is only able to see in the direction it is moving in. The projSpeed parameter is the speed at which the projectiles need to be travelling

```

def fov(self, screen, map, player):
    allWalls = map.getWalls()
    allWalls.append(player)
    projSpeed = 20
    if len(self.sightProj) == 0:
        self.createSightProjectile()
        print(self.sightProj)

    self.setProjDirection(projSpeed)
    for proj in self.sightProj:
        if proj.getCollided() == True or proj.getLaunched() == False:
            proj.setCollided(False)
            collided = proj.launchSightProjectile(screen, self.getHitbox().center, allWalls)
            print(proj.getPlayerCollision())
    self.setSeen(collided)

```

This is the fov method and it takes the parameters screen – this is a surface object, map – this is a map object and player – this is the player object. First a method is used on the map object in order to get all of the walls, then the player object is appended to this walls list for use later as the hunter needs to check whether the projectile hits the player or walls. Then the projSpeed variable is set as 20 and a check is done to see if the sightProj list is empty – this is so that we can check whether projectiles need to be created or if they have already been created. Next the setProjDirection method is used which is passed the projSpeed and sets the x and y speed for all 3 of the projectiles. Then a loop is used to check through each projectile and if the projectile has already collided with either a wall or the player or the projectile has not been launched yet then the collided attribute for the projectile is set to false in order to reset the projectile and then a method is used on the projectile in order to launch it which returns whether it has collided with the player or not – this value is only True if the projectile has collided with the player and not the wall.

Review:

In this iteration I have created the field of view for the hunter, however, I believe I should have done this after the pathfinding as the field of view depends on the direction in which the hunter is moving so it was slightly harder to test it in this circumstance. I also learnt that I should think more broadly when I create classes which need to be used in multiple different ways, in this case I needed to use the sight projectiles class in a slightly different way to the original one so I had to create a new one in order for the hunter to use it. This took some time and could have been resolved before if I made the sight projectiles class also suited to the hunter. Now that the 2 main aspects of the hunter are complete, the final part is the pathfinding for the hunter.

Specs completed/removed:

- P20 – As the hunter's FOV has now been added this has been completed
- P7, O10 – I decided to remove these criteria as I believe that I will not have enough time to do these and they are not useful for the game

- P24, O12, P25 – I decided to remove these from the game as I believe it would be unfair for the player to have a limit on their sprinting capabilities since the map is small so it would be a lot harder for the player to escape the hunter

Specs ongoing:

- P7, P12, P18, O6, O10, P19, P8, O9, P4, P5

Specs started:

- P14

Version 6.3:

In this version I am going to be coding the pathfinding part of the hunter so that the hunter is able to travel from where it is on the map to wherever the end coordinates are.

First I added some things to the object class which I created a while ago because I needed nodes for the A* algorithm and the classes for all of the objects which I have already created work perfectly to hold this part of the program

```
self.hCost = 0
self.fCost = 0
self.gCost = 0
self.last = None
self.path = False
```

These are the new attributes which I added to the object class. The hCost will be used for the heuristic cost of that node to the end coordinates which will be the Euclidean distance. Then there is an attribute for the fCost which will be the gCost and the hCost added together once both have been found. Then there is the gCost which is a measure of how far the node is from the start. The last attribute will be used to store the previous node which is in the path. Finally, the path attribute is just for testing so that I can change the colour of a node if it is part of a path.

```
def pythagoras(self, a, b):
    sqra = a**2
    sqrb = b**2
    sqrc = sqra + sqrb
    c = math.sqrt(sqrc)
    c = round(c)
    return c
```

This is the first new method added to the object class. It takes 2 parameters, a and b, these parameters will be integers and it does pythagoras on them, this will be used to calculate the Euclidean distance from this node to the end node.

```
def calcHCost(self, endCoords):
    startCoords = self.getCoords()
    xDist = endCoords[0] - startCoords[0]
    yDist = endCoords[1] - startCoords[1]
    dist = self.pythagoras(xDist, yDist)
    self.hCost = dist
    print(self.hCost)
```

This is the calcHCost method which takes the endCoords as a parameter – this is a tuple containing 2 integers for the x and y coordinate of the ending coordinates. First, since the path always originates from the hunter, the start coordinates are the coordinates of the hunter. Then the variables xDist and yDist are the difference between the x coordinates and y coordinates of the start and end points. Next the pythagoras method is used in order to calculate the Euclidean distance and is passed the x and y distances. Finally, the hCost for this node is set as the outcome of that.

```
def setFCost(self):
    self.fCost = self.gCost + self.hCost

def setGCost(self, value):
    self.gCost = value

def setGFCost(self, value = None):
    if value != None:
        self.setGCost(0)
    else:
        last = self.last
        self.setGCost(last.getGCost() + 1)
    self.setFCost()
```

These are the set methods for the rest of the cost attributes. The setFCost method just adds the gCost and the hCost and then sets the outcome as the fCost. The setGCost method takes a value and sets this as the gCost. The setGFCost method does both the previous methods in one but also includes a condition for the start node. This is because the start node has no previous nodes so the gCost has to be 0. If there is a value passed in then the gCost is calculated as being the gCost of the previous node plus one. Then finally, after that the fCost can be calculated – this has to be done at the end because the fCost cannot be calculated without the gCost or the hCost.

```

def getGCost(self):
    return self.gCost

def getFCost(self):
    return self.fCost

def getHCost(self):
    return self.hCost

```

These are the get methods for each of the cost attributes and they all return their respective attributes.

The rest of the changes below are done within the hunter class.

```
self.firstMove = False
```

I had to add this attribute to the hunter class because when the hunter is first spawned into the game then their position on the map is based on the coordinates of the top left of the rect which means that for the hunter to be centered on a node, I would have to first adjust it by a little the first time then I would be able to use the same movement for the rest.

```

def calcHCost(self, map, endCoords):
    allFloors = map.getFloors()
    for floor in allFloors:
        floor.reset()
        floor.calcHCost(endCoords)

```

This is the calcHCost method which takes parameters map and endCoords. Map is the map object and endCoords is the coordinates of where the hunter needs to end up.

```

#This is the code for the A* algorithm for the hunter to use
def aStar(self, endCoords, map):
    #This variable is for whether the point has been found
    found = False
    #These are the open and closed lists for the nodes
    open = []
    closed = []
    #The start node is wherever the hunter is
    startCoords = self.getMapCoords()
    startNode = map.getObject(startCoords)
    #This method is run in order to determine the H cost for every node to the endpoint
    self.calcHCost(map, endCoords)
    #As the start node is no distance from itself, the gCost for it is 0
    startNode.setGFCost(0)
    #The start node is added to the open list as its the first one to be considered
    open.append(startNode)
    print(open)

```

This is the first section of the A* method which takes the parameters endCoords – a tuple containing the x and y coordinates of the end location and map which is the map object. First a variable called found is initiated as false – This will be used in an indefinite loop to keep looking for a path until it has been found later on. Then there is the open and closed lists – these lists will be used in order to keep track of which nodes we need to look at, which node we are currently looking at and which nodes we have already looked at. The open list will contain the nodes we need to look at and the closed list will contain the nodes we have already looked at. The startCoords variable is the coordinates of the start node which will be the coordinates of wherever the hunter is on the map – a method is used to get these coordinates. Then another method is used on the map object in order to get the object itself which is at that location in the map – this is given the identifier startNode. Then a method is used in order to calculate the hCost of every traversable node in the map – this method is passed the map object and the coordinates of the end node. Next a method is used to set the gCost of the start node as 0 and to calculate the fCost. Then this first node is appended to the open list as we need to check it. The print statement was for testing.

```
#This loop runs while the endpoint has not been found
while found == False:
    #print('doingB')
    #Current is the lowest f cost in the open list,
    #This must be a high number so that it does not interfere with actual f costs
    lowestF = 10000000
    current = None
    #This loops through each element in the open list
    for x in range(len(open)):
        #print('doingO')
        #Checks if the f cost of the current node is greater than any other nodes in the open
        #list and if it is then the node with the lower f cost is set as the current
        if open[x].getFCost() < lowestF:
            current = open[x]
            lowestF = current.getFCost()
            index = x
    #After the node with the lowest fCost is found then the node is added to the closed list
    closed.append(current)
    print(current)
    #and deleted from the open list
    del open[index]
    #print(open)
    #print(closed)

    #This checks if the current node is the node which we are looking for
    #If it is then the loop will be broken
    if current.getCoords() == endCoords:
        print(current.getCoords())
        found = True
```

This is the second section of the A* algorithm method, this loop runs while the end node has not been found. First some variables are initialised, the lowestF is a variable which will be used to hold the value of the lowest f cost the a node in the open list and current will be used for the current node which we will be looking at. A for loop is used which goes through each of the nodes in the open list, then it checks

whether the fCost of the current element in the open list is lower than the lowest fCost we have found so far. If it is, then the current node is set as the current element in the list and the lowest fCost is set as the fCost of that node, then the index of the node is also held in a variable called index for later. As we have found the node which we will be currently looking at, we can remove the node from the open list and add it to the closed list so that we won't look over it again. First the current node is appended to the closed list and then it is deleted from the open list using the index of the element. After this we check if the current node is the end node, if it is then it will break the while loop by changing the value of found from false to true.

```
#This goes through each of the neighbours of the current node
for node in self.getNeighbours(current, map):
    #print('doingN')
    #This checks if the node is an object of type floor and that it is not already in closed
    if isinstance(node, objects.Floor) and self.checkClosed(closed, node):
        #This sets the last node for the neighbour node as the current
        node.setLast(current)
        #This sets the g and f cost of the node now based on the last node
        node.setGFCost()
        #This checks if the node
        open.append(node)
return current
```

This is the final section of the A* algorithm method. After checking whether the current node is the end node, if it isn't, then a for loop is run which goes through each node which is a neighbour of the current node. These neighbours are found using the getNeighbours method which takes the parameters current and map and returns a list of the neighbours of the node. Within the for loop, we first check whether the node is an object of type floor and that it is not already in the closed list, this is because the path cannot go through a wall and checking to make sure that the value is not already in the closed list is so that we do not go over the same node multiple times. If both of these are True then the previous node of the neighbour is set as the current node and the g and f cost of the node are set using the setGFCost method. Finally the neighbour is added to the open list to be looked at later. The return statement returns the current node when the while loop is broken which is when the end node has been found.

```
def findPath(self, endNode):
    #Base case - if g cost is 0 it must be the place where we began
    if endNode.getGCost() == 0:
        return [endNode]
    endNode.path = True
    list = self.findPath(endNode.getLast())
    list.append(endNode)
    return list
```

This is the findPath method which takes the endNode as a parameter which will be a type floor object. It is a recursive method with the base case checking when the gCost of the node passed in is 0 – this is because the gCost of the node is 0 at the node where the path originates from. If this is true then the

node is returned in a list. The path attribute for the node is set to True but this will only be used for testing purposes. Then the recursive line is run which is set as a variable which invokes itself with the previous node linked to the current node. Then after the list is returned, the current node is appended to the list and the list is returned.

```
#This method takes a node on the map and returns the neighbours of the node
def getNeighbours(self, node, map):
    coords = node.getCoords()
    cOne = (coords[0] - 1, coords[1])
    cTwo = (coords[0] + 1, coords[1])
    cThree = (coords[0], coords[1] - 1)
    cFour = (coords[0], coords[1] + 1)
    nOne = map.getObject(cOne)
    nTwo = map.getObject(cTwo)
    nThree = map.getObject(cThree)
    nFour = map.getObject(cFour)
    neighbours = [nOne, nTwo, nThree, nFour]
    return neighbours
```

This is the getNeighbours method which takes node – a floor object and map - a map object as parameters. First, a method is used to get the coordinates of the node which is passed as a parameter – this is given the identifier node. Then 4 variables are made which are tuples with the coordinates of the node but with the x or y coordinates changed by 1. There are 4 of these as the hunter is not able to move in diagonal directions, so the neighbours to a node would be above, left, right and below the current node. Then there are 4 more variables which hold the neighbour node – these are obtained using a method on the map object passed in and using the 4 different coordinates from before to get to get each one. Finally, all of these objects are added to a list called neighbours which is then returned.

```
def checkClosed(self, closed, current):
    for node in closed:
        if node == current:
            return False
    return True
```

This is the checkClosed method which is a linear search algorithm to look through the closed list and check whether each element of the array is the node it is looking for. If it is found, it returns true and if it isn't then it returns false.

```
def rearrangePathList(self, list):
    newList = []
    for x in range(len(list)):
        index = len(list) - x
        newList.append(list[index])
    return newList
```

This is the rearrangePathList method which takes the list of the path from the findPath method and rearranges it so that the start node is the first element of the list. I made this so it would be easier later on to make the hunter follow a path as it is easier to follow it when the list is in order. It first creates a new list called newList which is set as an empty list and then the for loop runs as many times as the number of elements in the list. The index variable is set as the length of the list subtract the number of iterations which the loop has done – this is so that the index goes backwards, then the element at the location of the index is appended to the newList variable. After all of the elements have been added to the newList variable, it is returned.

```
def traverse(self, node, map):
    if self.firstMove == False:
        loop = 4
        self.firstMove = True
    else:
        loop = 8
    hCoords = self.getHitbox().center
    gCoords = node.getRect().center
    xH, yH = hCoords
    xG, yG = gCoords
    xDir = xH - xG
    yDir = yH - yG
    print(hCoords)
    print(gCoords)
    if xDir > 0:
        for x in range(loop):
            self.moveLeft()
            self.setCoords()
            self.checkCollision(map)
    if xDir < 0:
        for x in range(loop):
            self.moveRight()
            self.setCoords()
            self.checkCollision(map)
    if yDir > 0:
        for x in range(loop):
            self.moveForward()
            self.setCoords()
            self.checkCollision(map)
    if yDir < 0:
        for x in range(loop):
            self.moveBackward()
            self.setCoords()
            self.checkCollision(map)
    print(self.getCoords())
```

This is the traverse method which takes a node and the map. The first thing that is checked is whether the firstMove attribute is false. This is so that we know whether it is the hunter's first movement in the

game. This is because when the hunter is first spawned into the game, its coordinates are not centered on a node as the top left coordinates for the hunter are used but the center of the node is used. This means that for the first move, the hunter only has to move half the distance which it usually has to, and since the size of each node on the screen is 32 pixels, this means that the hunter only has to move 16 pixels in both directions in order to be centered on a node. As the sprint speed of the hunter is set at 4 pixels this means that the hunter has to be moved in a direction 4 times for it to be centered on a node. Therefore, the loop variable is set as 4 as this is how many times the hunter will move 4 pixels in a direction. However, if it is not the first move then the hunter has to move the length of a full node each time to get from the center of one node to the center of the next node – as this is double the pixels the hunter needs to move – the number of loops for the hunter to move needs to be doubled as well. After setting the loop variable, the coordinates of the center of the hunter and the coordinates of the center of the node are obtained via different methods. Then each of the x and y coordinates in each of the coordinates are set as different variables. Then the difference between the x coordinates are calculated and then the difference between the y coordinates are also calculated. These variables will be used to determine which direction the hunter will move in. Then based on whether these values are positive or negative, the hunter will move in that direction. If the xDir is greater than 0 it means the hunter needs to move to the left and the opposite if it is less than 0. If the yDir is greater than 0 it means the hunter must need to move forward and the opposite if it is less than 0. For each decision, a for loop is run which loops as many times as the loop variable value and for each loop the hunter is moved, the coordinates of the hunter are set and the collisions are checked. This is because if I set the coordinates later on then the coordinates will be inaccurate and if I check for collisions later on then the hunter will have already moved a large distance before collisions can be checked. Which can cause the hunter to phase through the walls.

```
def pathfind(self, endCoords, map):
    endNode = self.aStar(endCoords, map)
    path = self.findPath(endNode)
    print('path:')
    print(path)
    self.setSprinting(True)
    for node in path:
        print('doing')
        self.traverse(node, map)
    self.setSprinting(False)
```

This the the overarching method which controls all of the elements for the hunter to pathfind – This method is passed the endCoords – destination coordinates and the map – map object as parameters. First a path is created using the aStar method which is passed the endCoords and the map and returns the final node. Then a path is created using the findPath method which returns a list for the path. When I coded this part of the algorithm then I realised that the list returned from the findPath algorithm does not need to be rearranged because as the recursion goes, the first element added to the list is the start node and then it appends the rest of the nodes from there. After the path is found, then the setSprinting method is used to set the sprinting value as True so that the sprintMultiplier is applied to the speed for the hunter and then a for loop is used to go through each node in the path and for each node the

traverse method is run so that the hunter can move to that node. Finally, at the end of the pathfind method, the sprinting attribute for the hunter is set to false.

Review:

In this iteration of the program, I have coded a basic version of the pathfinding which the hunter will use to traverse the map, however, it is a bit slow due to the for loops which are used to make the hunter traverse the map so I need to find a better way to implement the traversal for the hunter across the map. I also need to fix the errors where the hunter tries to walk through a few walls towards a node which it needs to traverse later on. I also learned that I need to make sure I think ahead about the whole program when coding classes as I could have made the object class more suited to the A* algorithm earlier when I coded the objects class, however, as I didn't I had to add things to it in this iteration.

Specs completed/removed:

- P16 – I have decided to remove this from the game as the map is quite small and the hunter can already easily hear the player so there is no need for the hunter to be alerted each time the player activates a lever

Specs ongoing:

- P7, P12, P18, O6, O10, P19, P8, O9, P4, P5, P14

Specs started:

- P3, P6, P11

Version 6.4:

In this version I am going to finish off the hunter's mechanics by improving some of the algorithms such as the sound algorithms and the pathfinding algorithm. I am also going to link all of the methods for pathfinding together.

```
self.path = None  
self.pathIndex = 0  
  
self.chasing = False
```

These are the new attributes which I have added to the hunter class the self.path attribute will be used to hold a list of the nodes which the hunter has to travel on. The pathIndex will be the index of the path list in order to keep track of which node the hunter should be on. Finally, the self.chasing attribute will be used to keep track of whether the hunter is currently chasing the player or not.

```

def traverse(self, node, map):
    '''if self.firstMove == False:
        loop = 4
        self.firstMove = True
    else:
        loop = 8'''
    hCoords = self.getHitbox().center
    gCoords = node.getRect().center
    xH, yH = hCoords
    xG, yG = gCoords
    xDir = xH - xG
    yDir = yH - yG
    #print(hCoords)
    #print(gCoords)
    if xDir > 0:
        #for x in range(loop):
        self.moveLeft()
        self.setCoords()
        self.checkCollision(map)
    if xDir < 0:
        #for x in range(loop):
        self.moveRight()
        self.setCoords()
        self.checkCollision(map)
    if yDir > 0:
        #for x in range(loop):
        self.moveForward()
        self.setCoords()
        self.checkCollision(map)
    if yDir < 0:
        #for x in range(loop):
        self.moveBackward()
        self.setCoords()
        self.checkCollision(map)

```

This is the new version of the traverse algorithm, I will get ride of the code in speech marks later on, however, now the way this method works is that it checks where the center of the hunter is in relation to the node it is aiming to travel to using the coordinates of the center of the hunter and the center of the node and then based on that, the hunter is then moved in a certain direction, the coordinates of the hunter are set and collisions are checked. I changed this because previously the loops would limit how often the program could register the player input and it also meant that the hunter would move the length of a full node before the player could move which made it unfair for the player.

```
#This is the overarching method for the pathfinding algorithm which includes the movement and the calculations
def pathfind(self, endCoords, map):
    endNode = self.aStar(endCoords, map)
    self.path = self.findPath(endNode)
    #print('path:')
    #print(self.path)
    self.setSprinting(True)
```

This is the pathfind method which takes the parameters endCoords – a tuple containing the end coordinates, and map – a map object. It first uses the method aStar in order to calculate the path to the end node, this is set as the variable endNode and then this is passed to the findPath algorithm which then returns a list of the nodes which the hunter needs to follow in order to reach the end node. Then it sets the sprinting for the hunter to True so that the hunter is able to move faster.

```
#This method checks whether the hunter has a path to follow or not
#and based on this it either makes the hunter follow that path or
#reset the variables for another path
def followPath(self, map):
    if self.pathIndex < len(self.path):
        node = self.path[self.pathIndex]
        self.traverse(node, map)
        nodeCoords = node.getRect().center
        if nodeCoords == self.getCoords():
            self.pathIndex += 1
            #print(self.pathIndex)
    else:
        self.path = None
        self.pathIndex = 0
```

This is the followPath method which takes the map object as a parameter. This method first checks if the attribute pathIndex is lower than the length of the list containing the paths to make sure that the hunter is not at the end of the path. Then if this is true – hunter is still following a path, then the node variable is set as the node at the value of pathIndex. This is then passed to the traverse method which takes the node and moves the hunter towards that node. Then the variable nodeCoords is set as the coordinates of the center of the node. A check is then done to see if the coordinates of the center of the hunter is the same as the center of the node. If it is then the pathIndex can be incremented as the hunter now needs to move onto the next node in the path. However, if the pathIndex is equal to the length of the path list then the path and pathIndex need to be reset as the hunter has reached the end of the path.

```
#This method checks whether the hunter needs to create a new path or not and
#it also checks if the hunter has either seen or heard the player.
def checkPath(self, map, player):
    hiding = player.getHiding()
    if self.path != None:
        if (self.heard == True or self.seen == True) and hiding == False and self.getChasing() == False:
            endCoords = player.getMapCoords()
            self.pathfind(endCoords, map)
            self.setChasing(True)
        elif hiding == True and self.path == None:
            self.setChasing(False)
            self.randomPath(map)
    else:
        self.setChasing(False)
        self.randomPath(map)
    self.followPath(map)
```

This is the checkPath method which, based on different situations, creates different paths for the hunter to follow. It takes the parameters map and player which are the map and player objects. First a method is used on the player object in order to get the hiding attribute for the player to check if the player is hiding. Then it checks to see whether there is already a path for the hunter to follow, if there is then it checks if it has heard or seen the hunter and that the player is not hiding and it is not already chasing the player – this last part is because if this was not added then the hunter would continually try creating new paths to track the player down. If this is True then it sets a variable for the end coordinates as the player's current position and then uses the pathfind method to create a path for the hunter to follow to that location. Then a method is used to set the chasing attribute to True so that the hunter does not keep recalculating paths to find the player. However, if the player is hiding then the hunter should only follow random paths in the map. As I was writing this, I realised an error in my code which is checking that the hiding variable is true and there is no path to follow, this would never run as the first decision made checks whether there is a path and this decision is made based on whether or not there is no path so this would never run. Continuing with the method, the else statement is if there is no path for the hunter to follow as they have reached the end of their path. It sets chasing to false and creates a random path for the hunter to follow. Finally, since there will always be a path after the above statements are run, the followPath method is run which makes the hunter follow a path

```
#This method generates a random path for the hunter to follow
def randomPath(self, map):
    floors = map.getFloors()
    index = random.randint(0, len(floors))
    floor = floors[index]
    endCoords = floor.getCoords()
    self.pathfind(endCoords, map)
```

This is the randomPath method which takes a map object as a parameter. First it gets all of the floor objects from the map. Then a random number is generated which is between the start of the floor list (0) and the end of the floors list which would be the length of floors, however, I have realised another mistake as the length of floors would be greater than the index of the floors array as the length starts

from 1 whereas indexing starts from 0. Continuing with the method, a floor is selected using the random number as and index for the list and a method is used in order to get the coordinates of that floor. This is set as the end coordinates. Finally the pathfind method is run and is passed the endCoords variable and map object which calculates a path to that coordinate.

```
#This method checks if the hunter has won the game
def checkWin(self, player):
    pRect = player.getHitbox()
    hRect = self.getHitbox()
    if hRect.colliderect(pRect) and player.getHiding() == False:
        return True
```

This is the checkWin method which takes a player object as a parameter and then uses a method on the player to get the hitbox for the player. then another method is used on the hunter in order to get the hunter's hitbox. Finally, a pygame method is used on the hitboxes to check if they collide while the player is not hiding. If this happens true is returned which will be used in the game loop to take the user to the lose screen.

```
def getChasing(self):
    return self.chasing

def setChasing(self, value):
    self.chasing = value
```

These are the get and set methods for the chasing attribute which return and set the chasing attribute.

```
def ready(self, screen, map, player):
    self.setCoords()
    #if self.visible == True:
    self.displayHunter(screen)
    self.sound(player, screen, map)
    self.fov(screen, map, player)
    self.checkPath(map, player)
    win = self.checkWin(player)
    if win == True:
        return True
```

This is the ready method, it is the overarching method which links all of the hunter's mechanics together - it takes the parameters screen, map and player which are the respective objects. First the coordinates of the hunter is set, otherwise the hunter would appear in the top left of the screen which would break the game – this will be used in conjunction with an if statement which is currently commented out for testing purposes, this would be so that the player can only see the hunter when the hunter is in the line of sight of the player. Then a method is used to display the hunter on the corresponding place on the screen, next the sound method is run which runs all of the sound mechanics for the hunter, then the fov method is run which controls all of the sight part of the hunters mechanics. Then the checkPath method is run which controls all of the pathfinding for the hunter. Finally, the checkWin method is run in order to check if the hunter has caught the player, a value is only returned when the hunter has caught the player, this value is True.

Review:

In this iteration I have now completed all of the hunters pathfinding aspects and fixed some of the bugs from the previous version such as the hunter walking through multiple walls to get to the path it needs to follow. However, a few new issues are now present in the game where the hunter is stuck colliding with a wall at a specific location or just walks through 1 wall at the start of the pathfinding. These will be fixed in the next iteration and the collisions will also be improved. Furthermore, I tested the hunter's field of view again to make sure it works along with the pathfinding as I mentioned when I created the field of view for the hunter and as it does not work as intended again, I will also have to fix this in the next version.

Specs completed/removed:

- P3, P11 – These have been completed as the random pathfinding has been implemented for the hunter so the hunter is able to randomly travel around the map.

- P14 – I have decided to remove this as I believe it is unfair for the player since the map is so small
- P5, P7 – As the hunter is able to know follow the player around the map when the player is seen, this is completed, however, I changed it slightly as I don't think that the hearing should be disabled so that the game is still realistic. Furthermore, I do not think that the red overlay is a necessity to the game
- P4 - This has been completed in the ready method
- P8 – I have decided to remove this as I believe that it would not make any impact on the game since the chances of the hunter hearing the player is already the correct amount, if I implemented this, it would mean the hunter would be able to hear the player all the time when sprinting
- P6 – I have completed this as the hunter is able to now chase after the player, however, I have decided to keep the speed of the hunter the same throughout the game as it creates problems when it changes and I believe the hunter is already fast enough
- P22 – This has been completed as the hunter can catch the player when they collide into each other which makes the player lose the game

Specs ongoing:

- P12, P18, O6, O10, P19, O9

Specs started:

- None

Version 6.5:

In this sub version I am going to fix all of the bugs from the testing above such as the hunter's field of view not working in the correct direction or how the hunter sometimes constantly collides into the wall in some situations

```

def fov(self, screen, map, player):
    allWalls = map.getWalls()
    allWalls.append(player)
    projSpeed = 20
    if len(self.sightProj) == 0:
        self.createSightProjectile()

    self.setProjDirection(projSpeed)
    collided = False
    for proj in self.sightProj:
        if proj.getCollided() == True or proj.getLaunched() == False:
            proj.setCollided(False)
            check = proj.launchSightProjectile(screen, self.getHitbox().center, allWalls)
            if check == True:
                collided = True
    self.setSeen(collided)

```

I have changed this method for the hunter. Previously the value which was set as the seen attribute would only depend on the third projectile in the list due to the way I had coded it which mean that it would only set the seen attribute as True if the third projectile in the list had hit the player which is wrong. However, now I have changed the identifier for the value returned from the launchSightProjectile method from collided to check and now collided is a separate variable. Now collided is set as false before any of the projectiles are checked, then if any of the projectiles have collided with the player collided is set as true, otherwise it is unchanged. Therefore, if any of the projectiles hit the player, the collided value will be set as True and seen will be set as True.

```

def pathfind(self, endCoords, map):
    endNode = self.aStar(endCoords, map)
    self.path = self.findPath(endNode)
    self.pathIndex = 0

```

I decided not to implement the sprinting mechanic of the hunter as it would cause problems when the hunter's speed changed midway through a path and crashed the game. Furthermore, the hunter was too fast when it was sprinting and I thought it would not be possible for the player to react in time due to the limited field of view of the player if the hunter was that fast. However, I also added setting the pathIndex attribute to 0 in this method as I realised this needs to be reset for each path so it would be more logical to set it to 0 here.

```
def followPath(self, map):
    if self.pathIndex < len(self.path):
        node = self.path[self.pathIndex]
        self.traverse(node, map)
        nodeCoords = node.getRect().center
        if nodeCoords == self.getCoords():
            self.pathIndex += 1
    else:
        self.path = None
```

This is the followPath method, I have just transferred teh resetting of the pathIndex attribute to the pathfind method above and removed the commented print statement,

```
def checkPath(self, map, player):
    hiding = player.getHiding()
    if self.path != None:
        if (self.heard == True or self.seen == True) and hiding == False and self.getChasing() == False:
            endCoords = player.getMapCoords()
            self.pathfind(endCoords, map)
            self.setChasing(True)
    else:
        self.setChasing(False)
        self.randomPath(map)
    self.followPath(map)
```

In the checkPath method, I have removed the unnecessary elif statement which did not work as it contradicted the previous decision above it.

```
def ready(self, screen, map, player):
    self.setCoords()
    if self.visible == True:
        self.displayHunter(screen)
    self.sound(player, screen, map)
    self.fov(screen, map, player)
    self.checkPath(map, player)
    win = self.checkWin(player)
    if win == True:
        return True
```

This is the ready method for the hunter, the commented out decisionis now back in the method and it makes sure that the hunter is only displayed on the screen when the hunter is within the field of view of the player.

Below are the changes which I made to the sprite class in order to fix some of the bugs within the game

```
def resetMovement(self):
    self.forward = False
    self.left = False
    self.right = False
    self.backward = False
```

This is the new resetMovement method which I have added to the sprite class. I decided to add this method to the sprite class as I realised that the main problem causing the hunter field of view to only point in 1 direction was the fact that the movement attributes were not reset in the player class, therefore, they all stayed as True when the program was running and the field of view for the hunter only pointed downwards (because the first direction checked for the hunter's movement is downward in the setProjDirection method). However, this method sets all of the movement attributes to false in the sprite class.

```
#This method moves the player up by a specific number of pixels in the negative Y direction
def moveForward(self):
    self.resetMovement()
    self.forward = True
    if self.sprinting == True:
        self.y -= self.speed * self.sprintMultiplier
    else:
        self.y -= self.speed
```

All of the movement methods in the sprite class have been changed in the same way as above. For each method, I have added the new resetMovement method when the method is run so that the movement value which is set as True is correct for the current situation.

```

def checkCollision(self, map):
    #This function returns all of the walls for the map in a list
    walls = map.getWalls()
    #This is how many pixels the player will bounce off the wall when collision occurs
    collisionBounce = 5
    #This loop goes through each wall in the list above
    for wall in walls:
        #Gets the rectangle of the wall
        wallRect = wall.getRect()
        spriteRect = self.hitbox
        #Checks if the rectangle of the player has collided with that rectangle (for the wall)
        if spriteRect.colliderect(wallRect):
            #Each of these if statements check which direction the player is currently moving
            #and based on this decision, the direction the player bounces back is determined
            if self.backward == True or abs(spriteRect.bottom - wallRect.top) <= collisionBounce:
                self.y -= collisionBounce

            if self.forward == True or abs(spriteRect.top - wallRect.bottom) <= collisionBounce:
                self.y += collisionBounce

            if self.right == True or abs(spriteRect.right - wallRect.left) <= collisionBounce:
                self.x -= collisionBounce

            if self.left == True or abs(spriteRect.left - wallRect.right) <= collisionBounce:
                self.x += collisionBounce

```

In the checkCollision method, in order to make collision bugs less frequent, I decided to implement an extra technique for collision detection based on the calculation of the coordinates of the sides of the sprite and the wall. It uses the absolute value of the difference of the coordinates of the 2 colliding sides for the wall and the sprite to help detect the side which the sprite is colliding into the wall from. The absolute value is compared to a variable called collisionBounce – this is the pixel tolerance for how close the sprite and wall can be before it is considered a collision. Based on this a coordinate value for the sprite is changed.

```

def getMapCoords(self):
    x = math.floor(self.hitbox.center[0]/32)
    y = math.floor(self.hitbox.center[1]/32)
    return (x, y)

```

The main problem with the path of the hunter being through walls and the hunter walking through walls is how I rounded the coordinates in the getMapCoords method in the sprite class. This is because of the ratio between the screen and the map. As 1 node on the map is equivalent to 32 x 32 pixels on the screen, when the sprite is on a node at the map, the coordinates used to calculate the map coordinates are the center of the sprite, as the center of the sprite is at the center of a node, it is 16 pixels off in each axial direction from the true position of the node on the map. For example, if the center of the hunter was at the center of the second node from the y axis, it would be 48 pixels away from the y axis.

Therefore, when it is divided by 32 in order to calculate the corresponding coordinates on the map, it is

always a whole number and a half, in this case the y coordinate would be 1.5 . As I was using the round function previously, this would mean that all the x and y values would be rounded up to a higher value and misrepresenting the hunter's location on the map which means in this case the hunter's y coordinate would be 2 and causes problems. This is because if a wall is next to the hunter in the positive x or y direction then it would mean that the game would think that the hunter is in the wall which is an invalid position for the hunter. Thus causing the hunter to collide with some walls constantly or walk through them.

Review:

In this iteration of the program I have learned to make sure I think thoroughly about the logic of my program before I implement it. This is because all of the bugs which I had in my previous version were very simple fixes, for example, the bug where the hunter sometimes went through a single wall or was constantly colliding with a wall was because of the way coordinates were rounded in the getMap method in the sprite class and the field of view bug was because I forgot to reset the movement attributes each time in the sprite class as well.

Specs completed/removed:

- P12, P18, O6, O9 – I have decided to remove this as I believe that these extra screens are not needed within my game and doesn't add to the complexity of the game
- P19 – I have decided to remove this from the game as I do not think that it is a necessary addition to the game and it would not be a good use of time to implement this
- O10 – I have decided to remove this as I think it would make the game too easy for the player if they knew how many levers they have left to find

Specs ongoing:

- None

Specs started:

- Non

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Evaluative Testing

In this section I will test all parts of the game together

Black Box testing table:

Shaded row meaning:

- Red – Failed
- Orange – Partial success
- Green – Success

V – Valid

I – Invalid

B – Borderline (BV – Borderline Valid, BI – Borderline Invalid)

Test No	Pic Ref	Ref	Pre-reqs	V/ I/ B	Input/Calc	Expected Outcome	Outcome
T1	S1, S2	i1 & O1	Player isn't hiding	V	'W'	Character faces North on screen and moves in that direction	Character moves up on the screen which is towards north
T2	S1, S3	i2 & O2	Player isn't hiding	V	'S'	Character faces South on screen and moves in that direction	Character moves down on screen which is south
T3	S1, S4	i3 & O3	Player isn't hiding	V	'A'	Character faces West on screen and moves in that direction	Character moves left on the screen which is west
T4	S1, S5	i4 & O4	Player isn't hiding	V	'D'	Character faces East on screen and moves in that direction	Character moves right on the screen which is east

T5	S1, S6, S7, S8, S9	i5 & O8	Player isn't hiding Player have enough sprint energy	V	Any directional button and 'Shift'	Character moves faster in whatever direction of that key and the sprint energy decreases	Character moves faster on the screen
T6	N/A	i5	Player has no sprint energy left	I	Any directional button and 'Shift'	Character should not sprint and should only move in whatever direction at normal speed	Player is able to sprint – This is because this was not implemented
T7	S10, S11, S12, S13	i6, O7, P13 & P15	Player is within activation radius of a hiding spot or a lever	V	'E'	Character should carry out whatever action depending on what they are standing close to	When character is close to the hiding spot and presses 'E' then they are able to hide. When the character is close to a lever and presses 'E' then it is activated
T8	S14, S15	i6, O7, P13 & P15	Player is outside activation radius of a hiding spot or a lever	I	'E'	Character should not do anything	When character is outside activation radius and presses 'E' for both the hiding space and the lever, nothing happens
T9	S16, S17, S18, S19	i6, P13 & P15	Player is on the edge of the activation radius of a hiding spot or lever	BV	'E'	Character should carry out whatever action depending on what they are standing close to	When character is on the edge of the activation radius for hiding space or lever and presses 'E' then the action is carried out

Name: Muqtasid Zayyan Dar

Project title: Manhunt

T10	N/A	i7	Player is in menu	V	Left clicks on start button with mouse	Should take the user to the difficulty screen	
T11	S20	i7	Player is in menu	I	Right clicks on start button with mouse	Should not do anything	When right mouse button is clicked while user is on menu screen then nothing happens
T12	S	P1	Player is in menu	V	Left clicks on the options button with mouse	Should take user to options menu with list of controls	When the user left clicks on the settings button in the bottom left of the screen then all of the controls are displayed as text on the screen
T13	N/A	P2	Player is in difficulty menu	V	Left clicks on either normal mode or nightmare mode	Should take user to a separate screen which has tips for the player and ends the screen once the map has loaded in	
T14	S	P4 & P5	Player is in the game	V	Player moves into the hunter's FOV	The hunter should see the player and start chasing them at a faster speed and moving directly towards the player	
T15	S	P4&P5	Player is in the game Player is outside hunter's FOV and hearing range	I		The hunter should not start chasing the player	

Name: Muqtasid Zayyan Dar

Project title: Manhunt

T16	S	P6 & P23	Player has been seen by the hunter	V	'Shift'	When the player starts sprinting, it should seem that the hunter is being left behind	
T17	N/A	P7 & Ae1	Player has been seen by the hunter	V		There should be some alert which tells the player that they have been seen by the hunter	
T18	S	P8	Player is moving	V		There should be a sound circle originating from the player as they are moving	
T19	N/A	P8	Player is sprinting	V	'Shift'	There should be a bigger sound circle compared to before	
T20	N/A	P10	Player's sound circle and hunter's sound circle overlap	V	%overlap divided by number of walls between the hunter and the player e.g. 30/1	Hunter should have a random chance of 0.3 to hear the player	
T21	N/A	P10	Player's sound circle and hunter's circle are on the border of one another	Bl		The hunter should not have any chance of hearing the player as there is no overlap	
T22	S	P11	Player is not hiding	V		The hunter should generate a random predicted path to follow and at the end of the predicted path the hunter	

Name: Muqtasid Zayyan Dar

Project title: Manhunt

			Player just left hunter's FOV			should return to its usual random paths	
T23	N/A	P14	Player is in hunter's FOV	V	Player presses 'E' to hide	Hunter catches player even while the player is in the hiding spot	
T24	S	P14	Player was in hunter's FOV	I	Player presses 'E' to hide	Hunter should start searching for the player and not catch the player in the hiding space	
T25	N/A	P15 & O10	Player is within activation radius of lever	V	'E'	The lever counter on the HUD should increment by one and the light next to the lever should turn green	
T26	S22	P17 & O9	Player has activated all of the levers	V		The escape door should be opened and there should be an alert informing that it has opened	When all of the levers are activated, the doors change colour to show they are unlocked.
T27	S21	P17	Player hasn't activated all of the levers	I		The escape door should remain closed	When the number of activated levers are less than the number of levers in the game then the doors stay the same colour and player cannot escape
T28	S	P18	Player has activated all of the levers	V	Player walks through the escape door	An end screen should show up informing the player they have escaped before returning the player to the menu	When player walks into the activated door then the end screen is show and informs the player that they have won.

Name: Muqtasid Zayyan Dar

Project title: Manhunt

T29	N/A	P19	User is on the difficulty screen	V	Left click on either normal or nightmare button	The relevant map and hunter should be loaded in	
T30	S	P21	Wall(s) between the player and the hunter	I		Hunter should just continue as normal	
T31	S	P22 & O6	Player is within the catch radius	V		Hunter should catch the player and losing screen should be displayed	
T32	S	P22 & O6	Player is outside the catch radius	I		Hunter should not catch the player	
T33	N/A	P24	Player is moving	V	'Shift'	Stamina bar on HUD in bottom right of screen should start decreasing	
T34	N/A	P24	Player isn't moving	I	'Shift'	Stamina bar shouldn't decrease	
T35	N/A	P25	Player isn't sprinting	V	Not Shift	Stamina bar should start replenishing	
T36	S1, S2	O5	Nothing in front of the player's character	V	'W'	Character should move	When there is nothing in front of the character, it is able to move across the screen
T37	S23, S24	O5	Wall in front of player's character	I	'W'	Character shouldn't move	When there is a wall in front of the player's character and they walk into it then they cannot move through it and bounce back
T38	S25	O11	Player is enclosed by	V		The player should be able to see everything	When the player is in the game, they are only able to see what is around

Name: Muqtasid Zayyan Dar

Project title: Manhunt

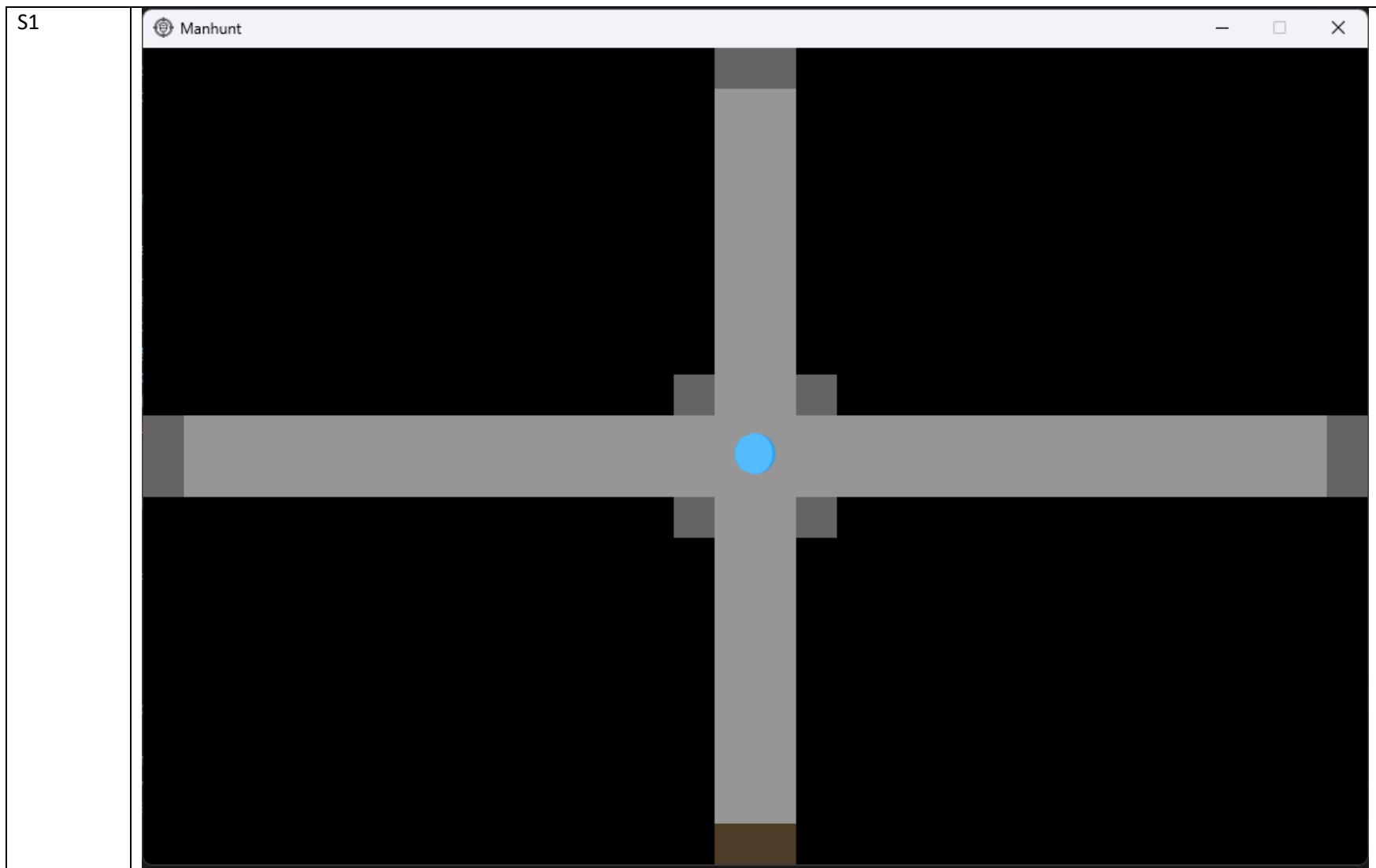
			walls around them			around them up to the walls	them up till the walls of the map, they are not able to see through the walls
T39	N/A	O12	Player stamina bar is low/empty	V		An alert informing the player to wait for the stamina bar to replenish should pop up	Removed form the game
T40	S						

Test Screenshots – need to update

Pic Ref	

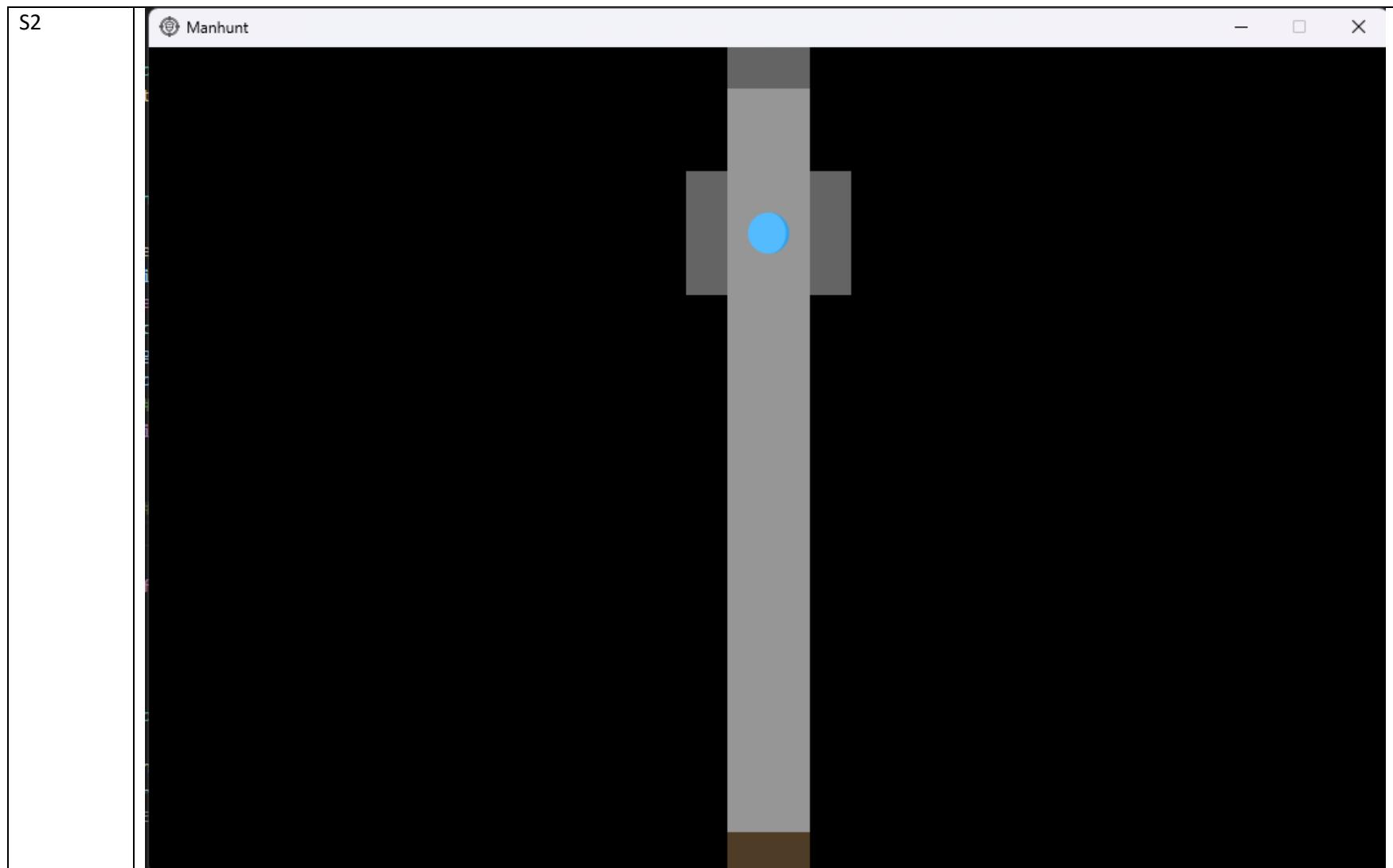
Name: Muqtasid Zayyan Dar

Project title: Manhunt



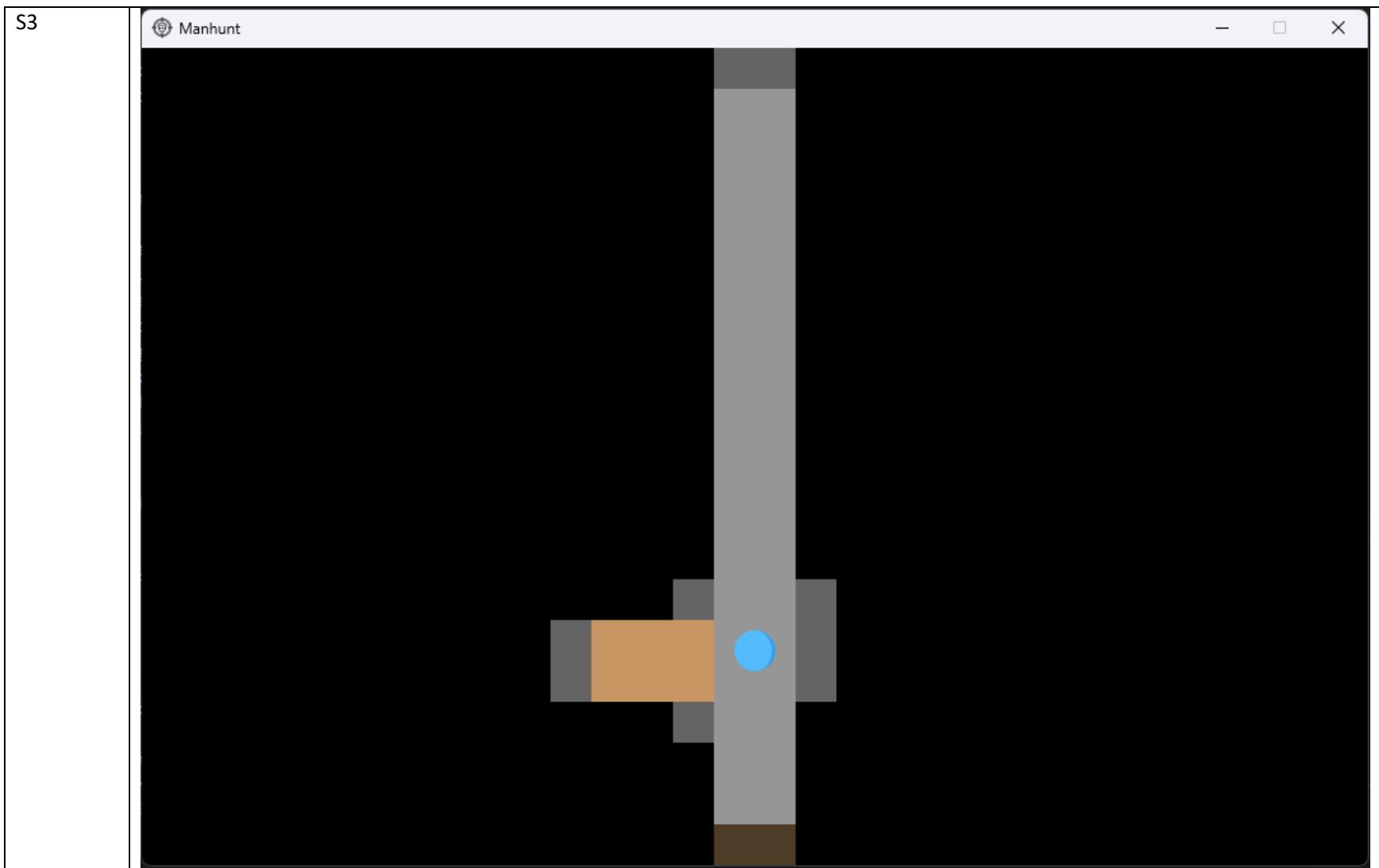
Name: Muqtasid Zayyan Dar

Project title: Manhunt



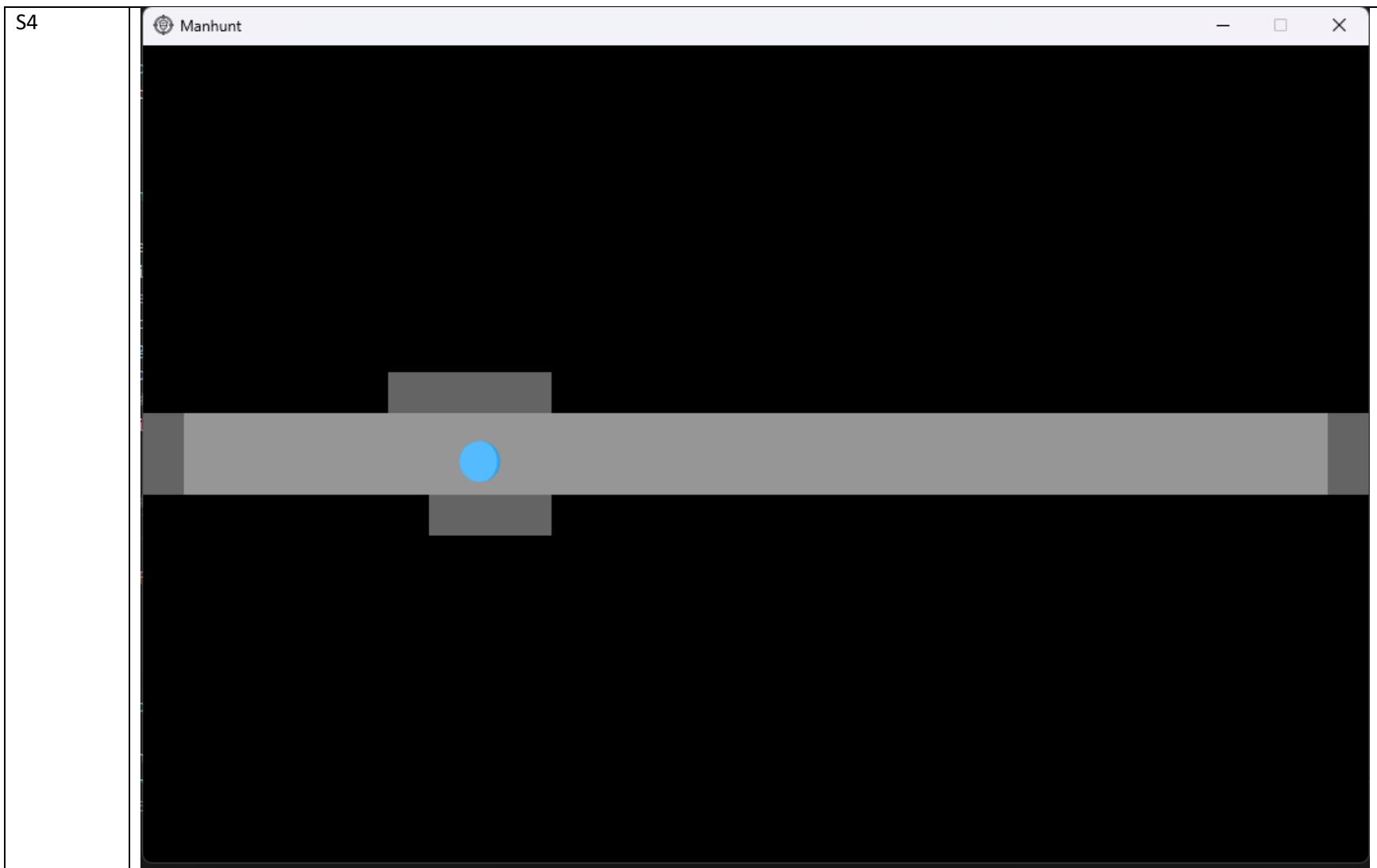
Name: Muqtasid Zayyan Dar

Project title: Manhunt



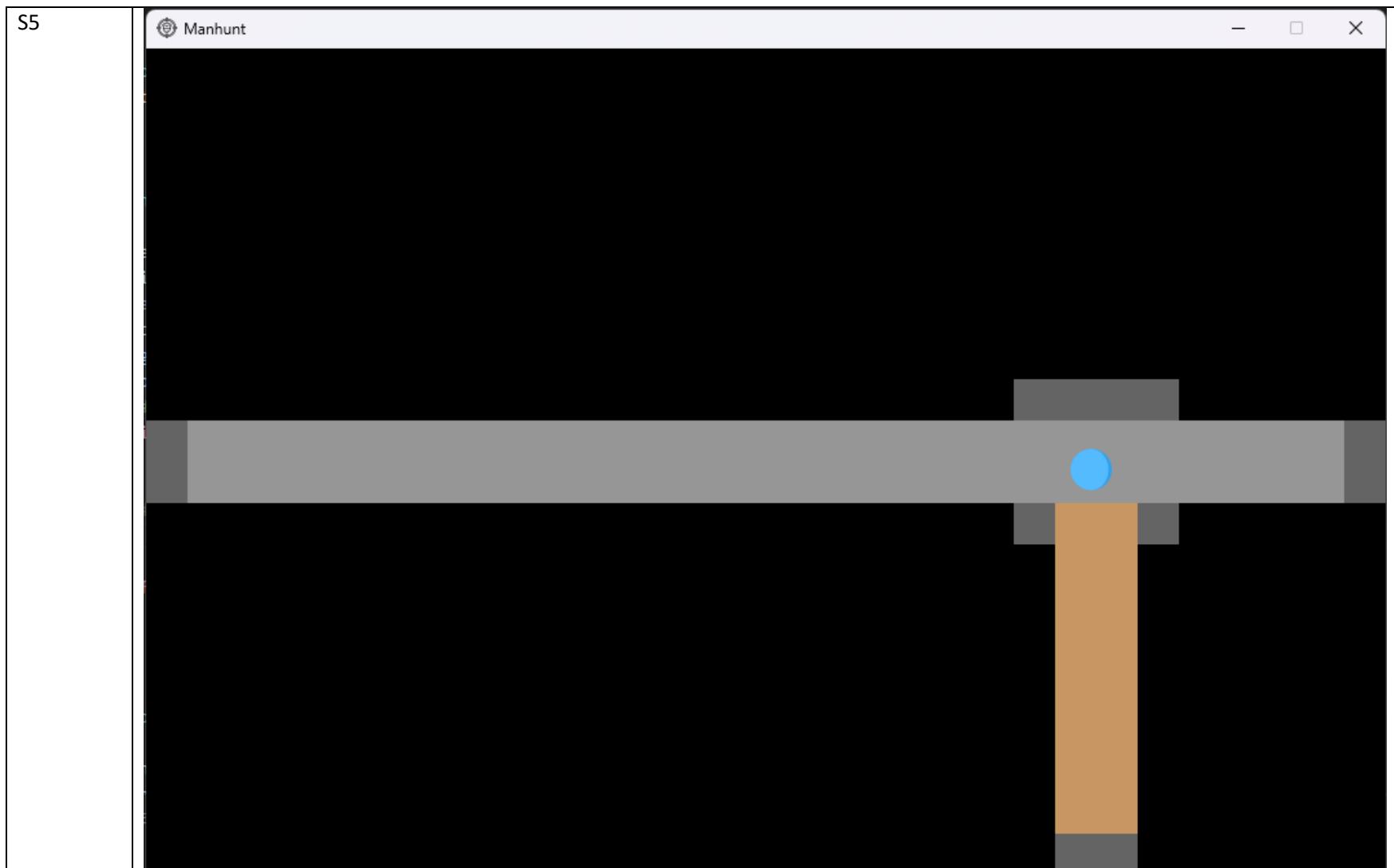
Name: Muqtasid Zayyan Dar

Project title: Manhunt



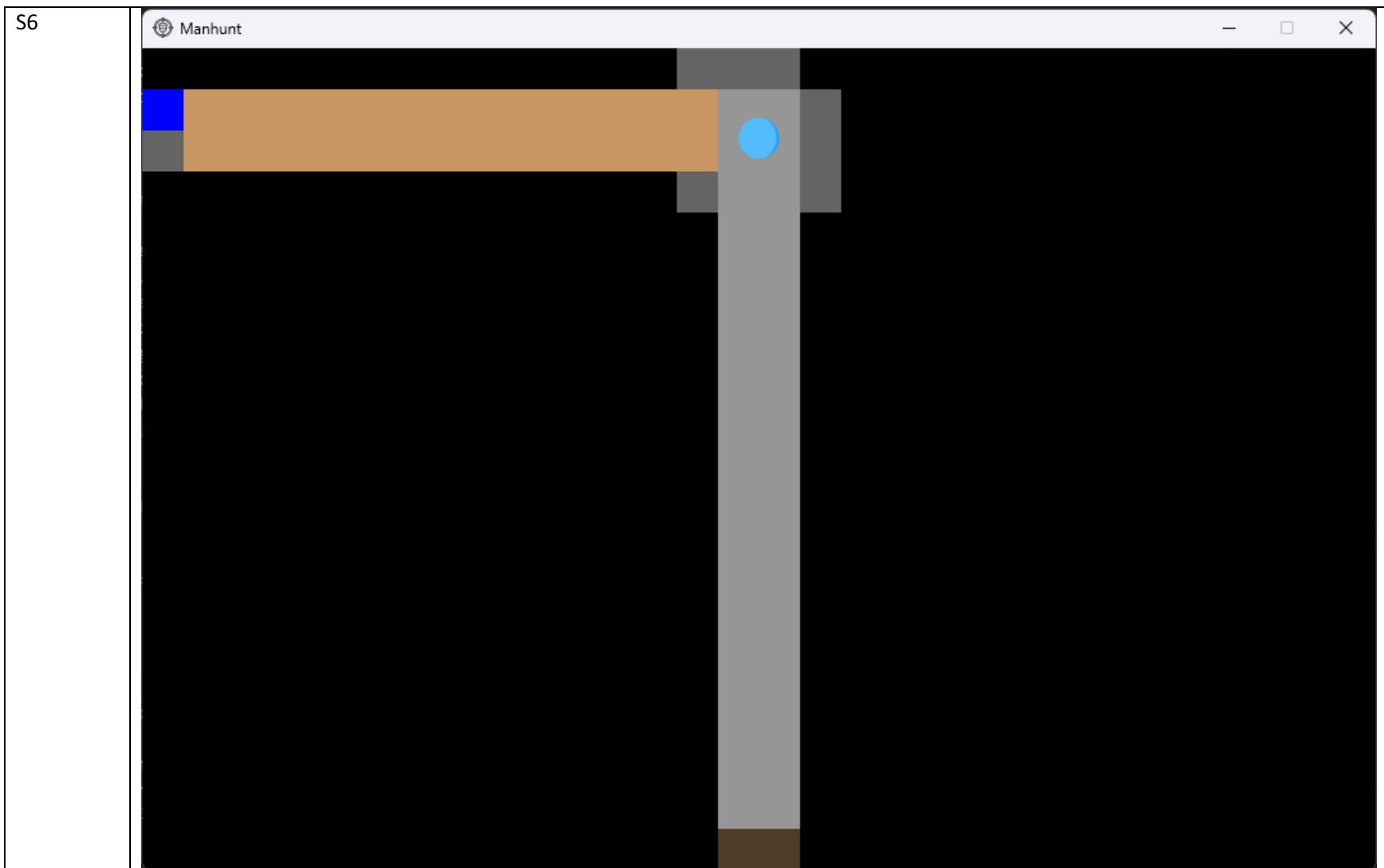
Name: Muqtasid Zayyan Dar

Project title: Manhunt



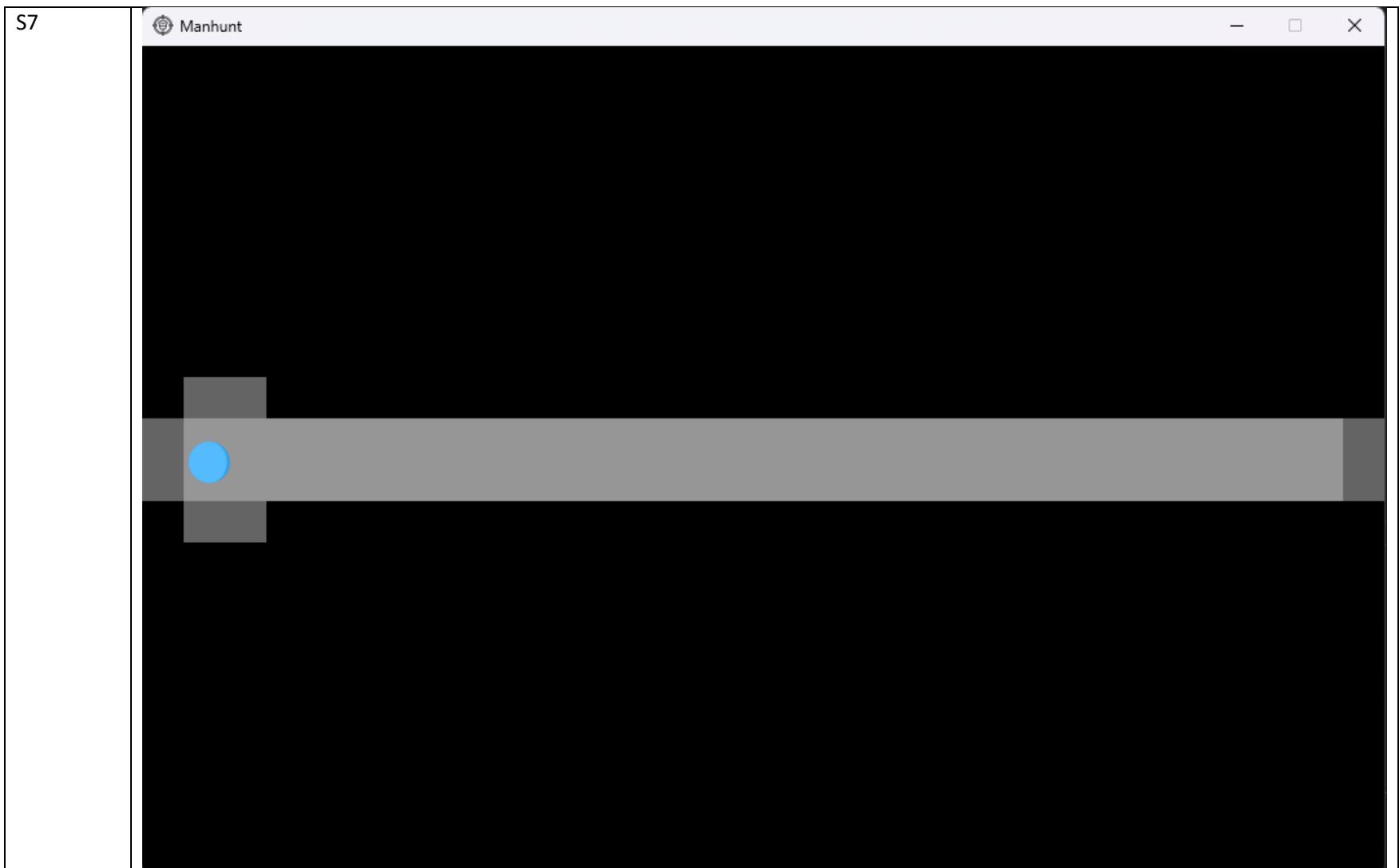
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



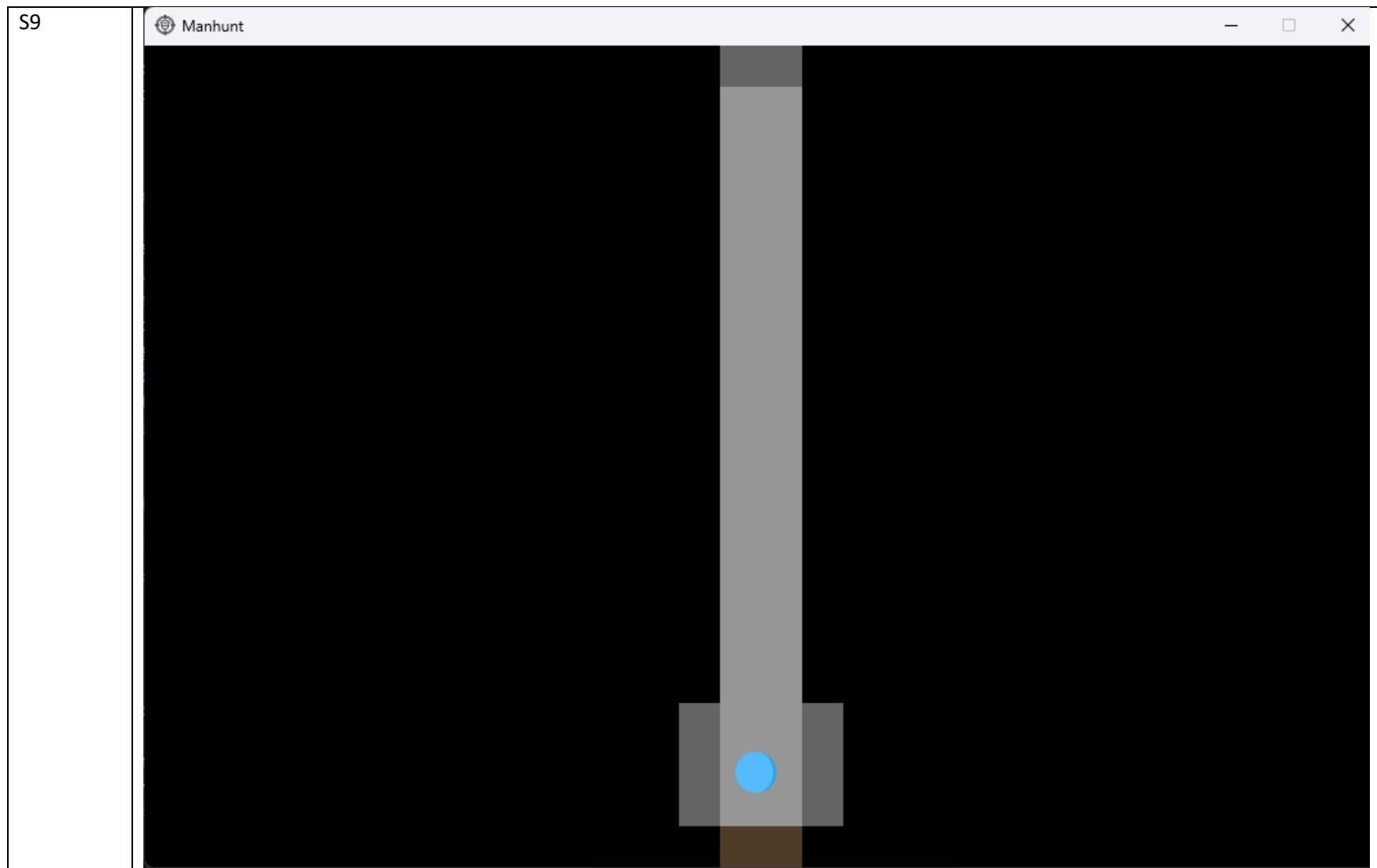
Name: Muqtasid Zayyan Dar

Project title: Manhunt



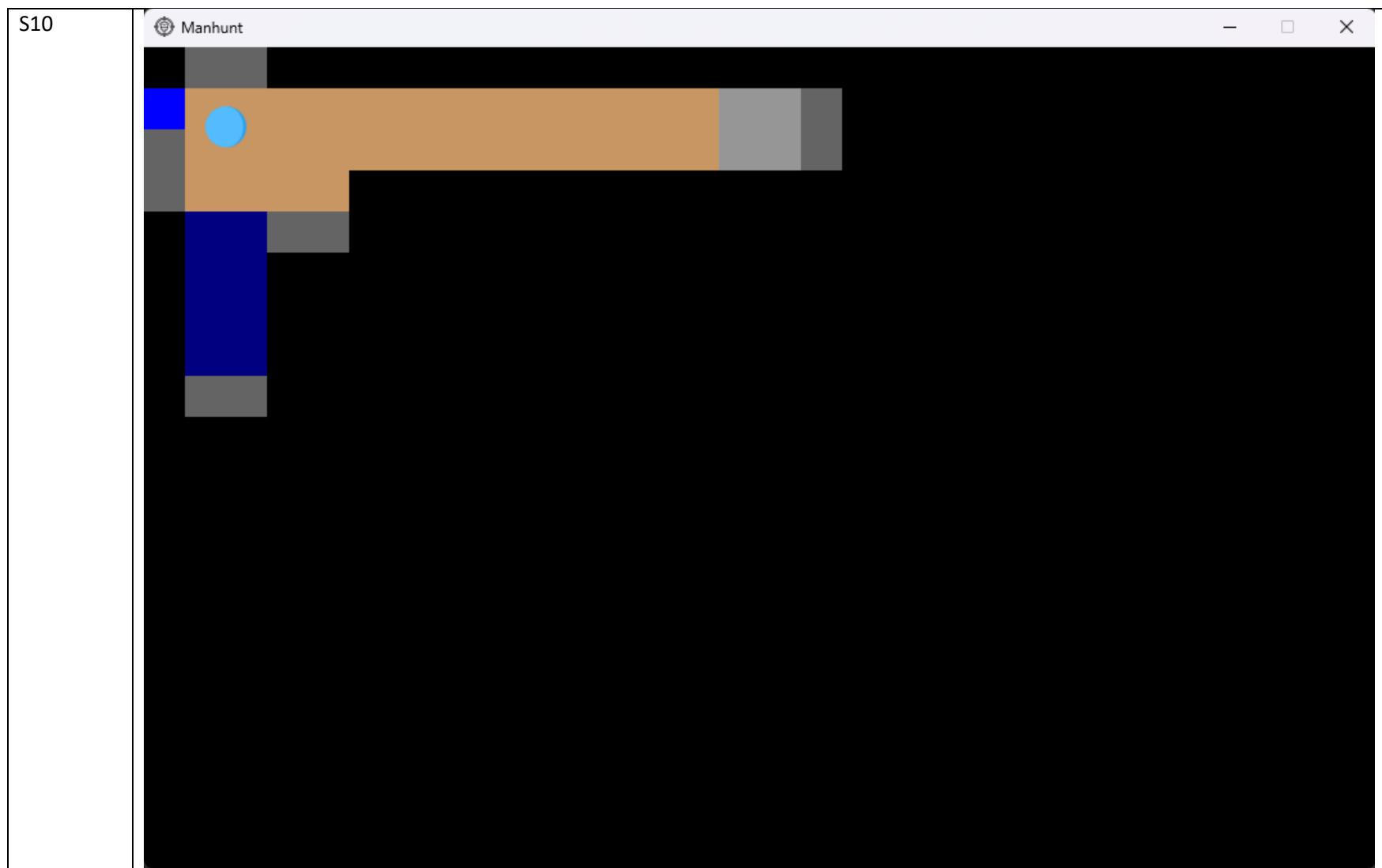
Name: Muqtasid Zayyan Dar

Project title: Manhunt



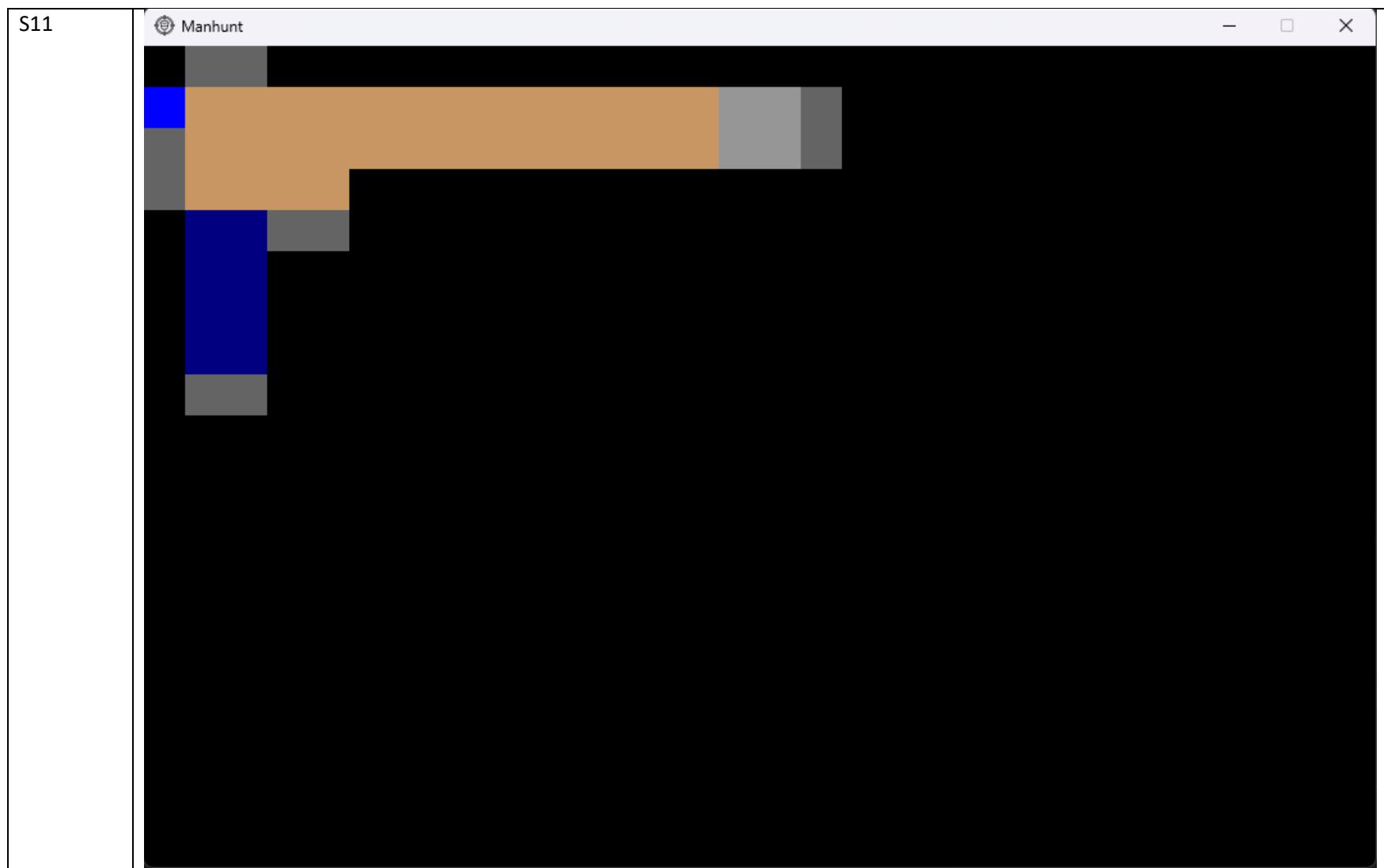
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



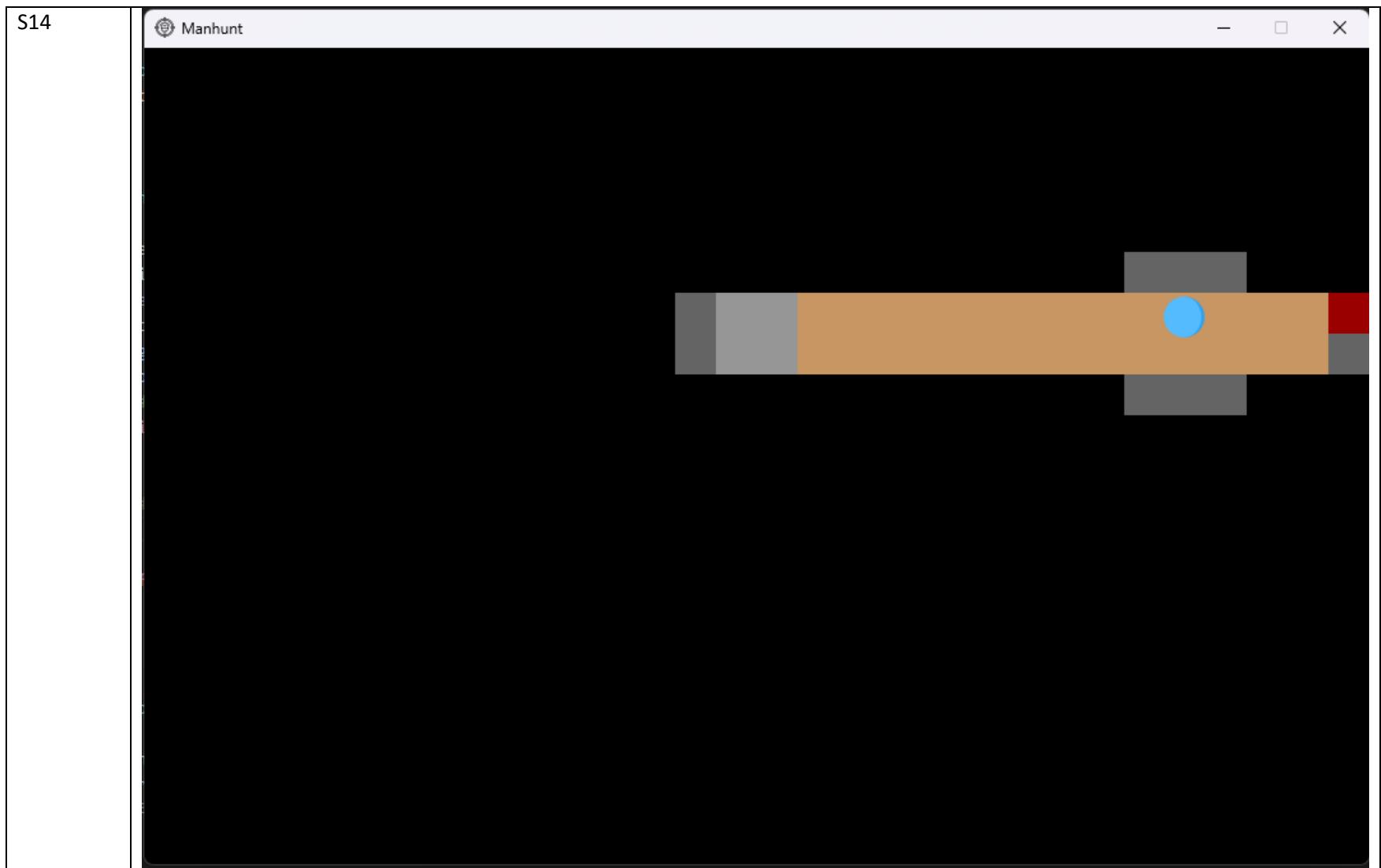
Name: Muqtasid Zayyan Dar

Project title: Manhunt



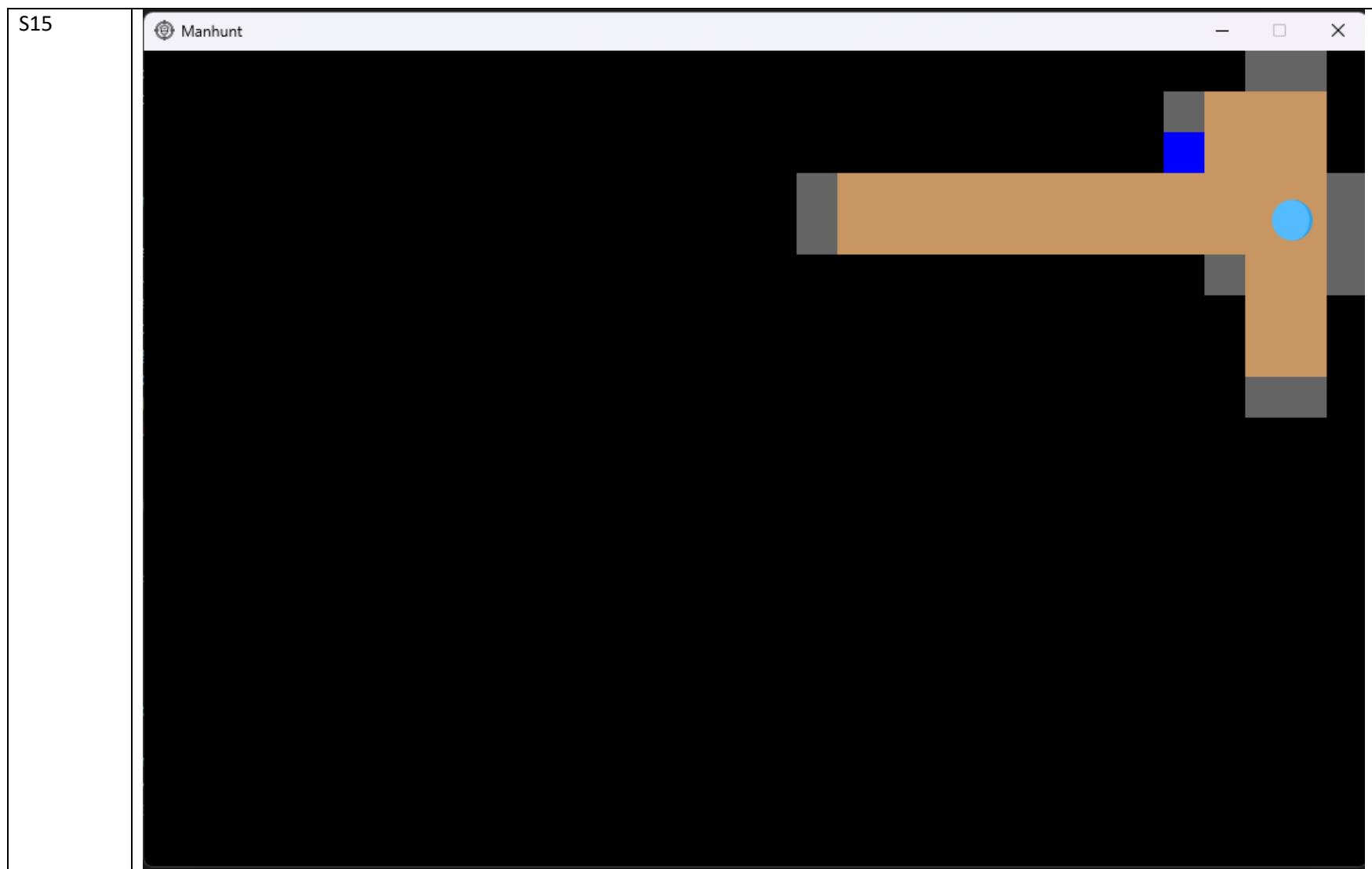
Name: Muqtasid Zayyan Dar

Project title: Manhunt



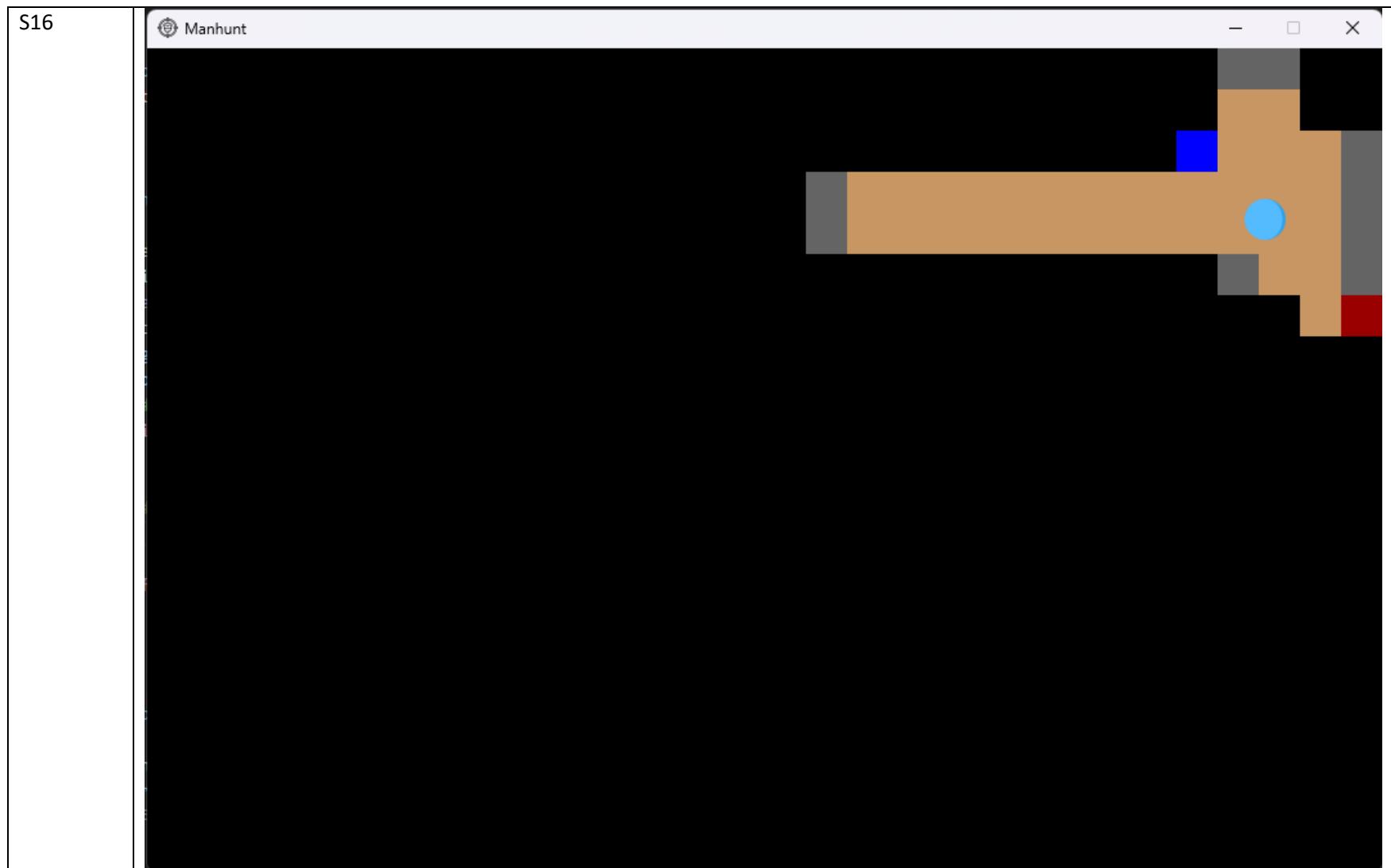
Name: Muqtasid Zayyan Dar

Project title: Manhunt



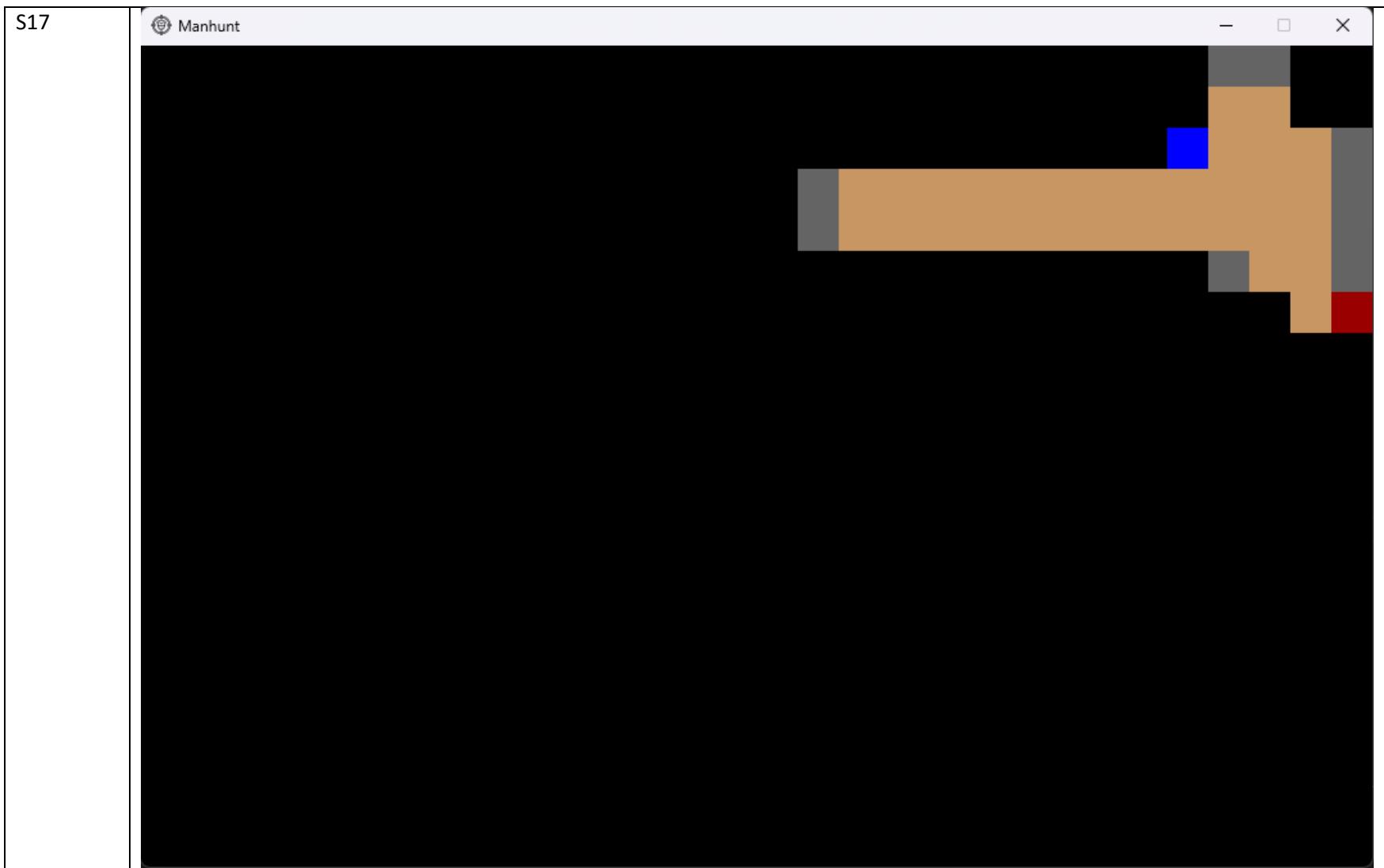
Name: Muqtasid Zayyan Dar

Project title: Manhunt



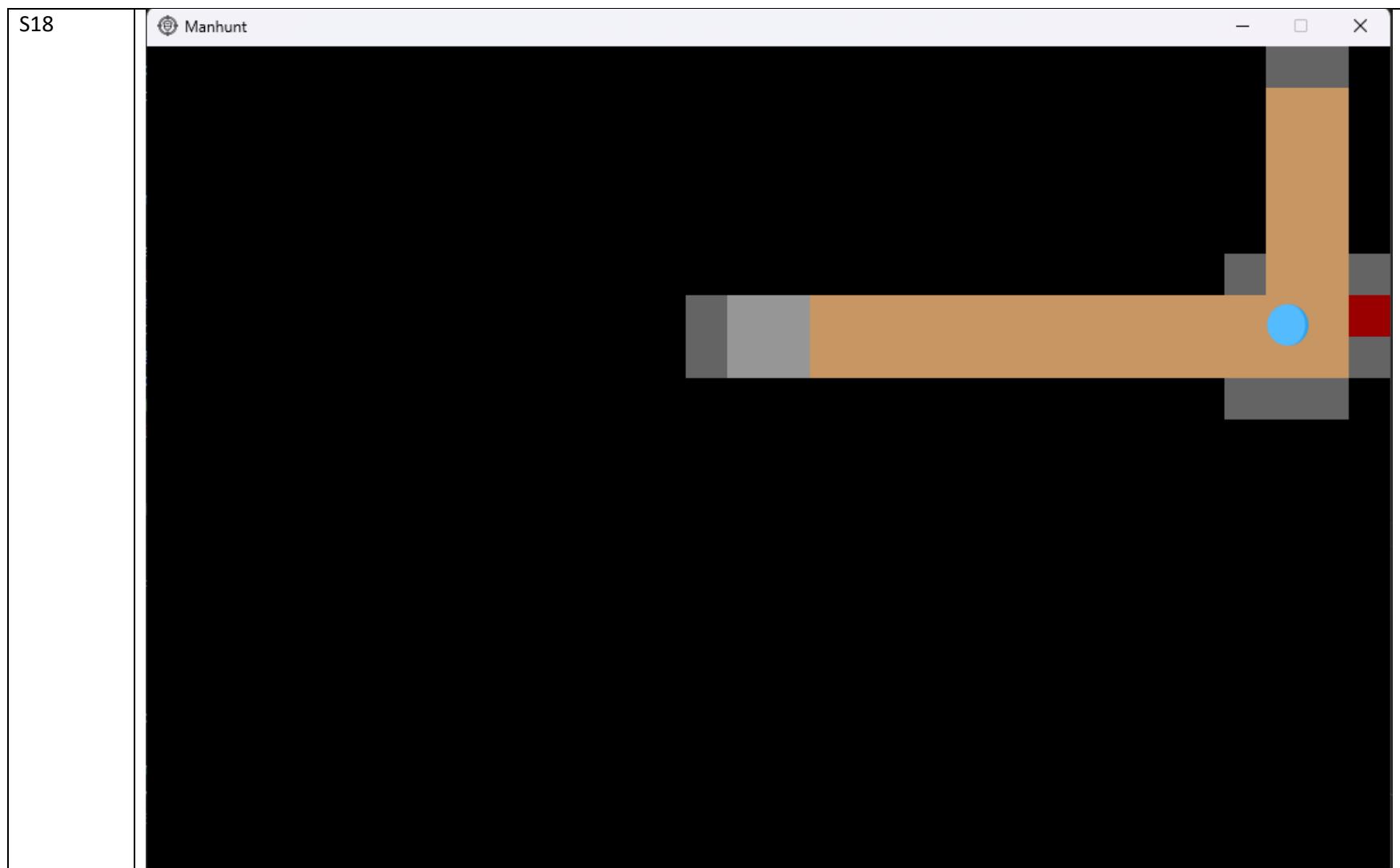
Name: Muqtasid Zayyan Dar

Project title: Manhunt



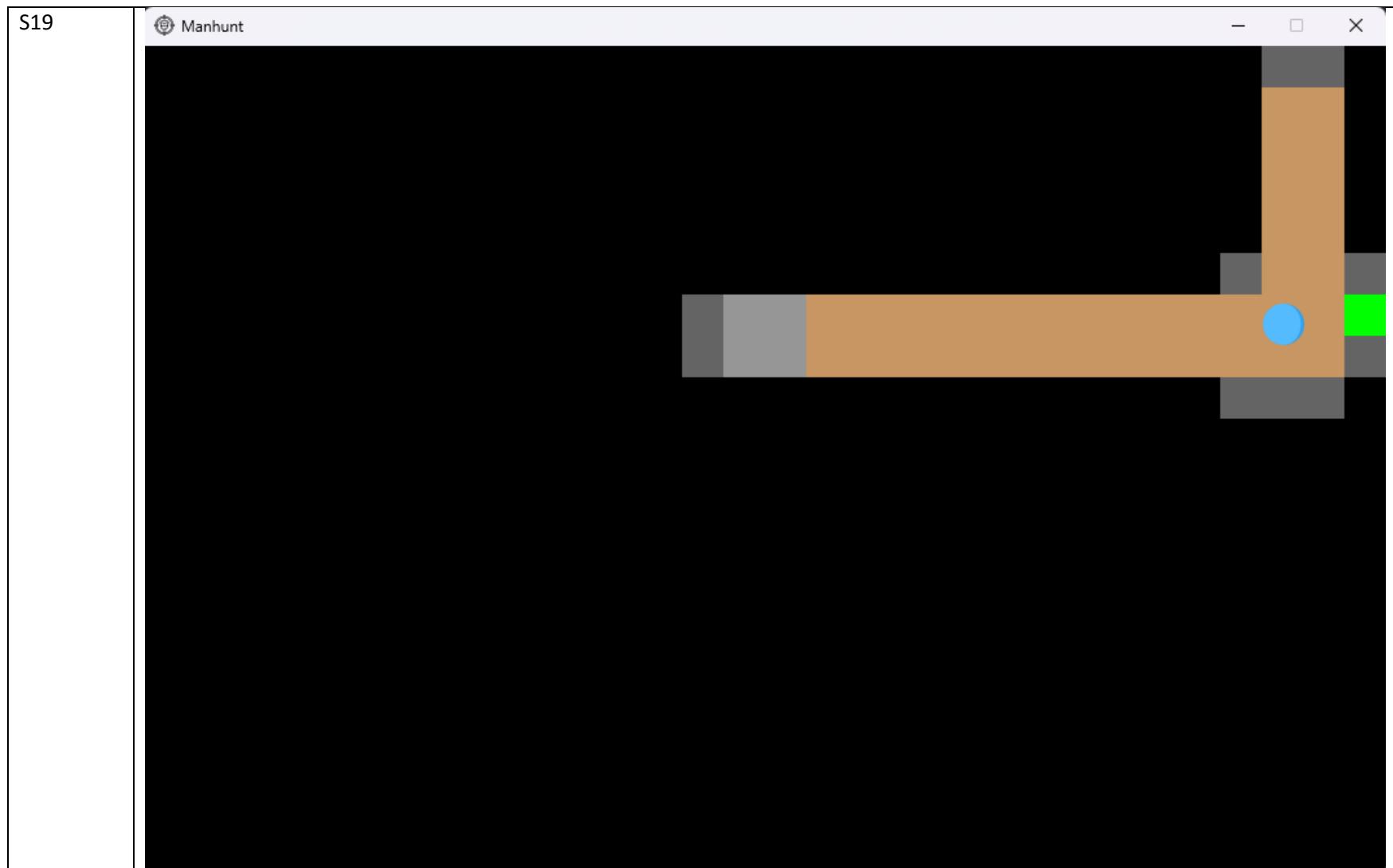
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



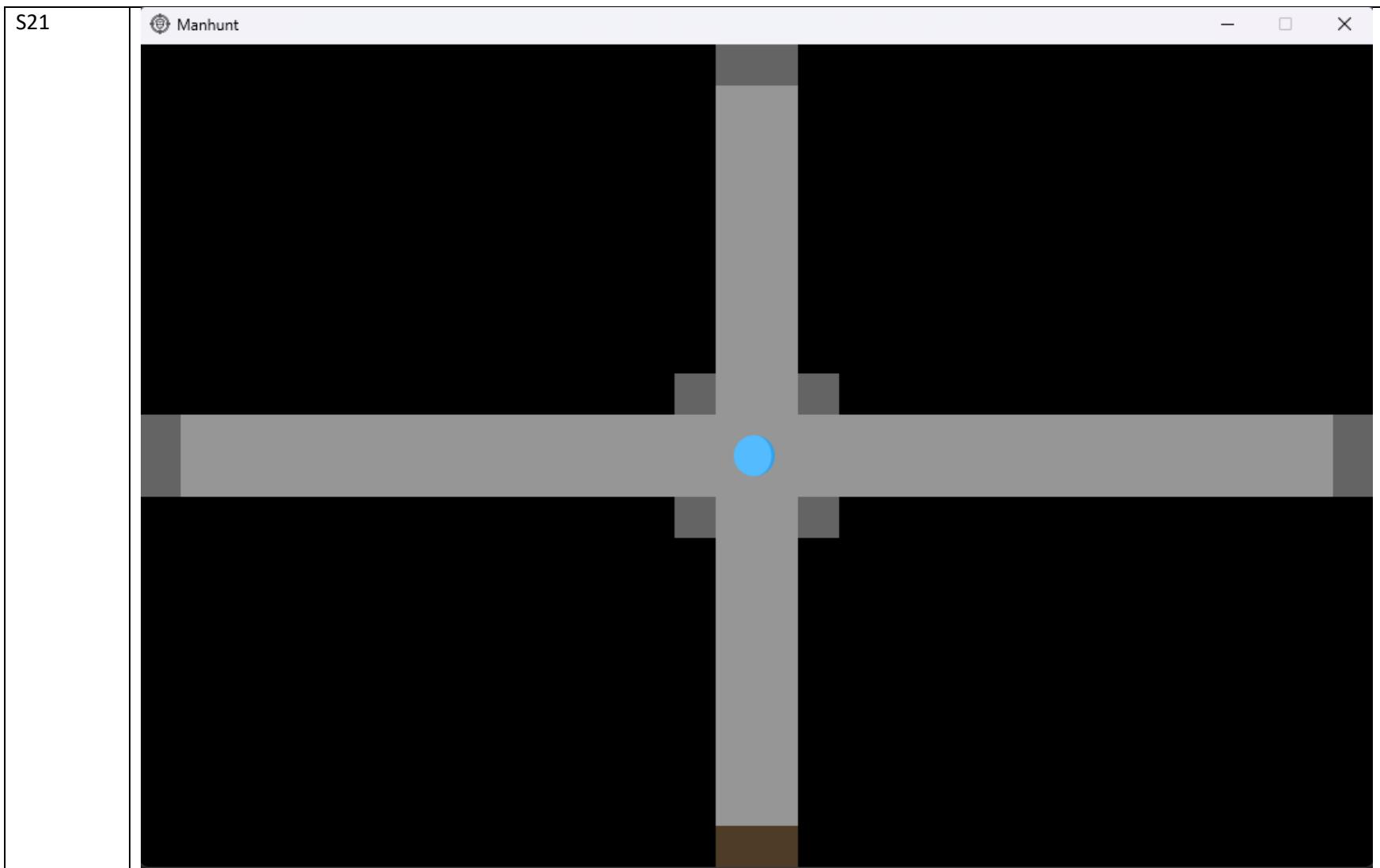
Name: Muqtasid Zayyan Dar

Project title: Manhunt



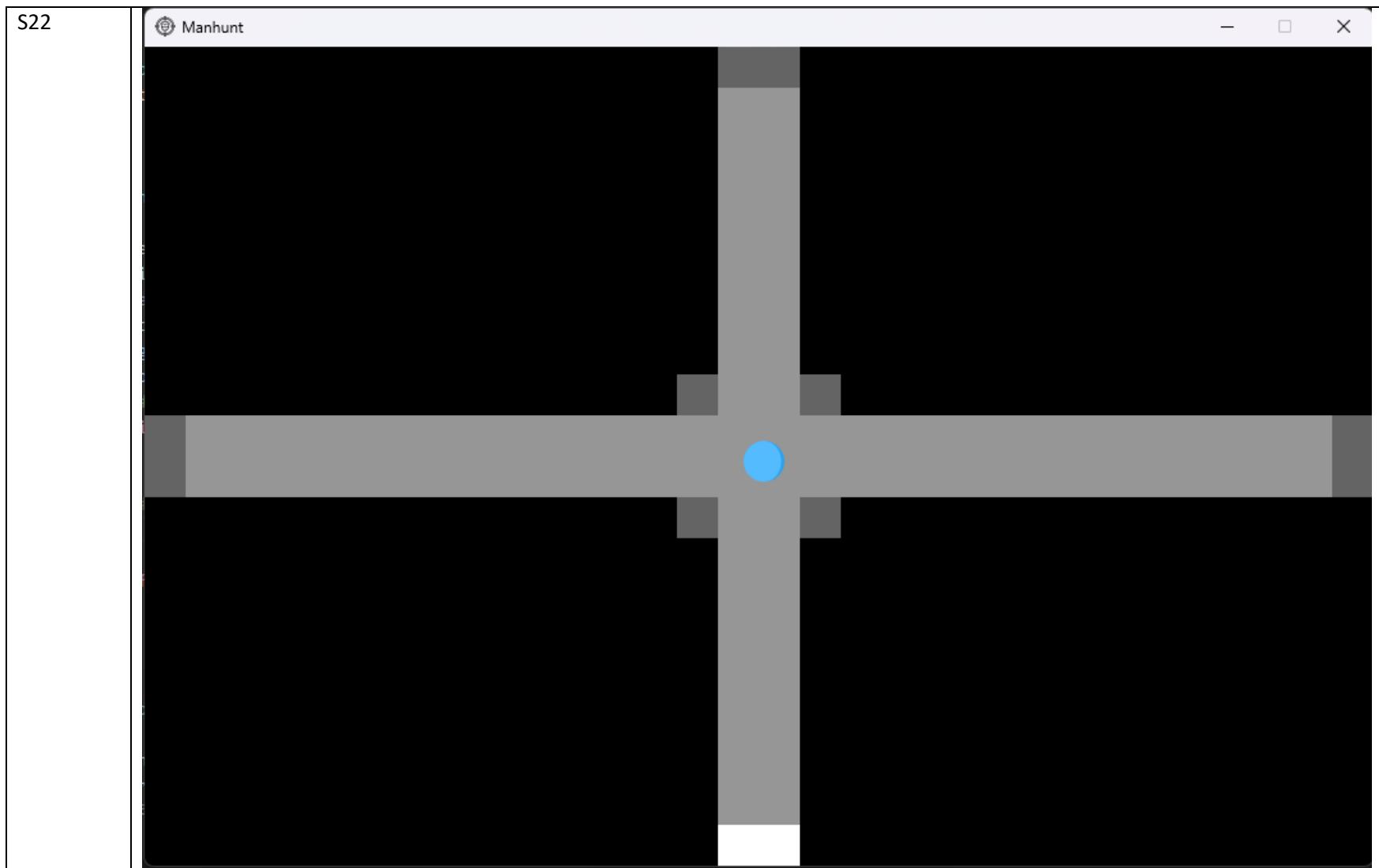
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



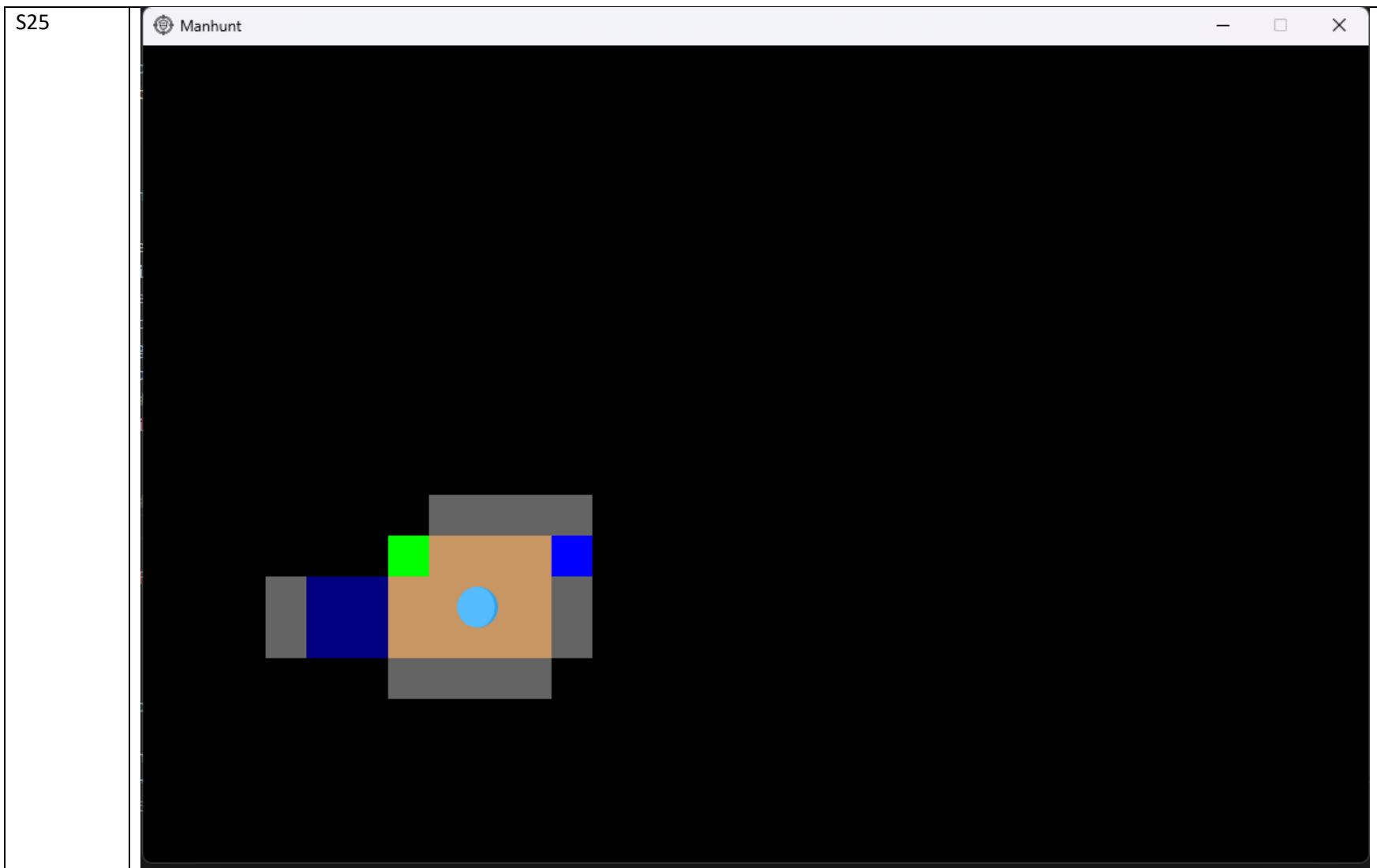
Name: Muqtasid Zayyan Dar

Project title: Manhunt



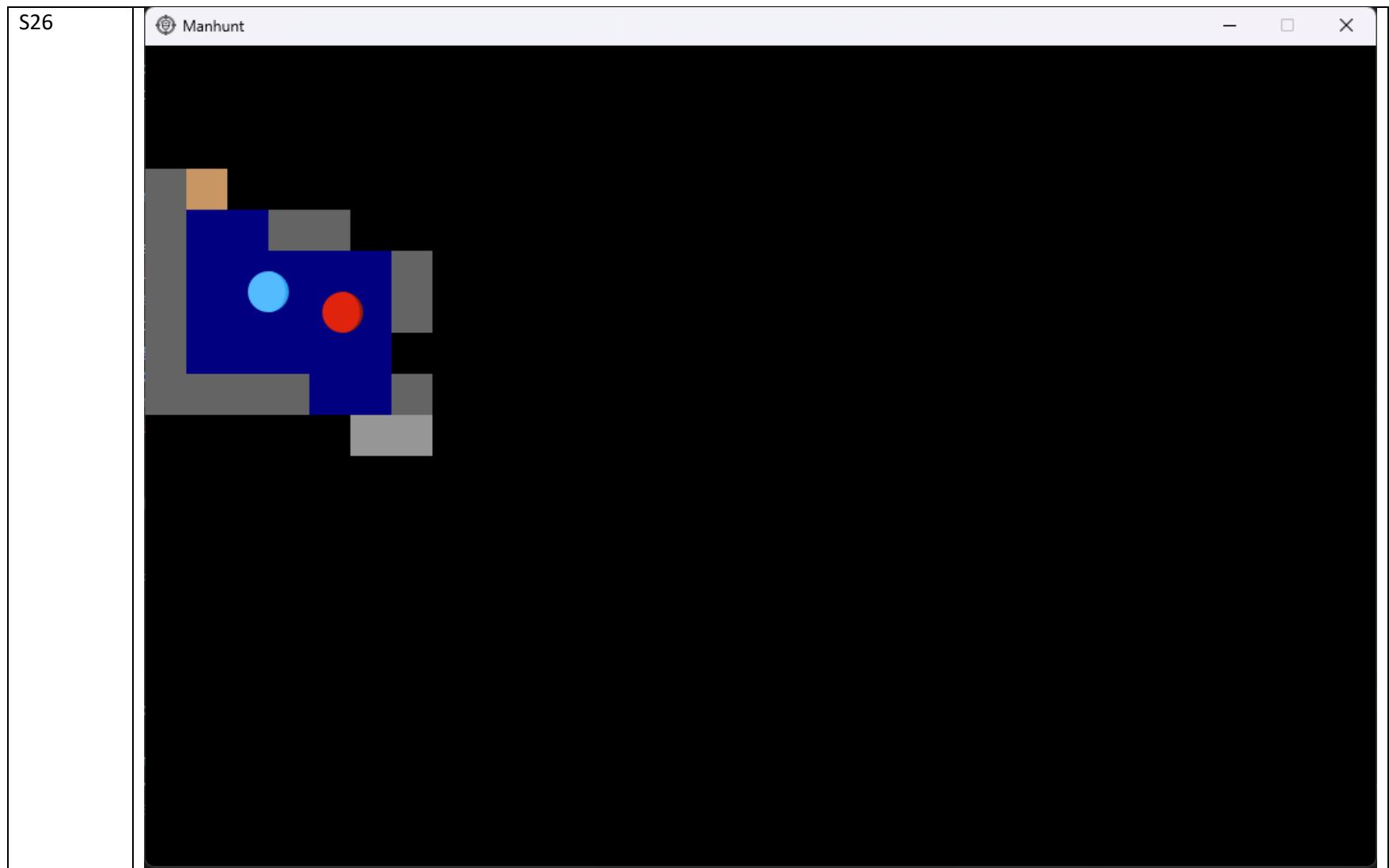
Name: Muqtasid Zayyan Dar

Project title: Manhunt



Name: Muqtasid Zayyan Dar

Project title: Manhunt



Stakeholder Testing:

Name: *Samer*

Test	Works? (✓ or ✗)	Comments?
Character moves when WASD keys are pressed	✓	
Character moves faster when any of the WASD keys are pressed with Shift key	✓	
Only able to see up till walls, not through the walls	✓	
When character moves into a wall, the character collides and bounces back	✓	Sometime player goes into the wall when sprinting
Mouse can be used to navigate the menu screens	✓	
When character is next to a hiding space then by pressing E key the character hides	✓	
When character is next to a lever then by pressing E the colour of the lever changes from red to green	✓	

Name: Sameer

Map resembles the inside of a building	<input checked="" type="checkbox"/>	Mostly, but not many familiar objects to make it look like the inside of a building set apart from the structure
Can escape from the hunter	<input checked="" type="checkbox"/>	
When hunter catches player then you lose	<input checked="" type="checkbox"/>	
When all levers are activated then the doors are unlocked	<input checked="" type="checkbox"/>	
When you walk through the door then you win	<input checked="" type="checkbox"/>	
When the hunter sees the player then the hunter starts chasing the player	<input checked="" type="checkbox"/>	On some occasions hunter doesn't start chasing player although it seems like hunter saw player
When the player is hiding and hunter sees the player, hunter does not start chasing the player	<input checked="" type="checkbox"/>	
When you click on the cog at the bottom left on the starting menu then it displays a list of the controls	<input checked="" type="checkbox"/>	

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Name: Sameer

When you run around the map then the hunter sometimes start chasing you	<input checked="" type="checkbox"/>	
Player has 360 degrees vision	<input checked="" type="checkbox"/>	Laggy in long hallways
When player is running and hunter is chasing player then the player is faster than the hunter	<input checked="" type="checkbox"/>	
Comments on game overall:		Game is how I expected but some of the user experience could be better such as including screens which tell the user whether they won or lost the game or the log when the player walks into a big space

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Name: shahir

Map resembles the inside of a building	✓
Can escape from the hunter	✓
When hunter catches player then you lose	✓
When all levers are activated then the doors are unlocked	✓
When you walk through the door then you win	✓
When the hunter sees the player then the hunter starts chasing the player	✓
When the player is hiding and hunter sees the player, hunter does not start chasing the player	✓
When you click on the cog at the bottom left on the starting menu then it displays a list of the controls	✓ Could also display what the different colours in the map mean as they may just look like different coloured squares to a new player

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Name: Shahir

Test	Works? (✓ or ✗)	Comments?
Character moves when WASD keys are pressed	✓	
Character moves faster when any of the WASD keys are pressed with Shift key	✓	
Only able to see up till walls, not through the walls	✓	
When character moves into a wall, the character collides and bounces back	✓	Sometimes player goes through walls when sprinting into a corner.
Mouse can be used to navigate the menu screens	✓	
When character is next to a hiding space then by pressing E key the character hides	✓	
When character is next to a lever then by pressing E the colour of the lever changes from red to green	✓	

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Name: Shahir

When you run around the map then the hunter sometimes start chasing you	✓	
Player has 360 degrees vision	✓	Sometimes parts of the room cannot be seen even though nothing is blocking the view. Laggy in big spaces
When player is running and hunter is chasing player then the player is faster than the hunter	✓	
Comments on game overall:		Game is mostly what I expected but some things in the game can be improved like the vision of the player or the collisions

Both stakeholders comment on the collisions in the game and say that the player sometimes goes through walls. Shahir specifies that it is usually in the corners which makes me think that this issue is most likely caused by the way the collisions was coded because when the player collides with 1 wall in the corner it could push the player back into another wall which would cause a collision and that wall would do the same thing, until the player phases through the wall and no more collisions occur. Sameer points out that the game does not look like it is set inside some sort of building due to a lack of 'familiar objects', although this would make the game look more like it occurs inside a building it would take time which I did not have and may take away from some of the other elements in the game which is why I did not implement it. He also says that sometimes the hunter looks like it is looking at the player but does not start chasing the player, I believe that this may be due to the player being just outside of the hunter's FOV at the end of the chase path created which means that the hunter then decides it cannot see the player and generates a random path. They also both specify that the game is a bit more laggy in the long hallways or bigger spaces in the game, I believe this is due to the amount of projectiles which are created in the long hallways which slows the game down when compared to a smaller room due to the amount of objects present. Sameer also states at the end that he would like the user experience to be better, however, due to time constraints I had to remove some of this as it did not add anything to the game itself and could removed some of the focus from the important aspects of the game.

Evaluation:

In this section I am going to assess how well different parts of the project went

Success Criteria Evaluation:

In this section I am going to evaluate my success criteria which I created in my analysis

Key:

Purple – Changed (Since analysis)

Green – Fully met

Orange – Partially met

Red – Not met

Spec Ref	Test Ref	Success Criteria	Evaluation
i1	T1	The player can press the 'W' button on the keyboard to move their character towards North in the level	I did this by creating a method in the sprite class which decreases the y coordinate of the sprite by a certain amount.
i2	T2	The player can press the 'S' button on the keyboard to move their character towards South in the level	I did this by creating a method in the sprite class which increases the y coordinate of the sprite by a certain amount.
i3	T3	The player can press the 'A' button on the keyboard to move their character towards West in the level	I did this by creating a method in the sprite class which decreases the x coordinate of the sprite by a certain amount.
i4	T4	The player can press the 'D' button on the keyboard to move their character towards the East in the level	I did this by creating a method in the sprite class which increases the x coordinate of the sprite by a certain amount.

Name: Muqtasid Zayyan Dar

Project title: Manhunt

i5	T5, T6	The player can press the 'Shift' button on the keyboard to move their character faster	I changed this success criterion as I believe it would be unfair to make the player wait to have to run from the hunter on such a small map where the hunter could easily sneak around the corner and catch up with the player
i6	T7, T8, T9	The player can press the 'E' button in certain locations to do an action	I accomplished this by having an attribute for the player class which is the area around it in which the player can interact with object. Then when the player presses the 'E' key a method within the player class takes all of the objects the player can interact with in the map and then checks through all of them to see if the area of the player has collided with that object which would mean that the player is close enough to the object to interact with it and then based on what the object is, another method is invoked which then carries out the action for that object
i7	T10, T11	The mouse will be used so that the user can use the start menu to play	For this I used a method from the pygame library which returned the location of the mouse pointer on the screen then, I used a method from the button class which I created in order to check if the mouse collided with the button. If it did and the user presses the left mouse button then the screen is changed to a different one.
P1	T2	On the start screen, there will be a controls button and a start button, when the controls button is pressed then the controls of the game will be displayed	I changed this success criterion so it would be more generalised as the controls screen which I have made has the controls displayed in a text format which the user can then read in order to understand what the controls for the game are. I did this by using the button class I created to create a button object which is a control button and then I displayed this in the bottom left corner of the main menu screen.
P2	N/A	When the player presses start then a loading screen will appear while the game is loaded in the background with the hunter and the player starting in different locations.	I decided to remove this from the success criteria as the game loads very quickly since it is not a very demanding game to load in, therefore, a loading screen would be pointless.
P3	T	A loop will immediately be in effect for the hunters AI so that it starts following randomly generated paths throughout the game.	I did this by creating a random path method which generates random paths for the hunter to follow in the map
P4	T	There will also be another loop within the hunter's Ai that will be in effect checking whether it can see or hear the player	I did this by creating a method which uses calculations and factors such as the number of walls between the player and the hunter to determine whether or not the hunter is able to hear the player.

Name: Muqtasid Zayyan Dar

Project title: Manhunt

P5	N/A	If the hunter hears the player then the hunter's speed should be increased (not as much as if it sees the player) and the shortest path to the location should be assigned for the hunter to follow. While the hunter is following the path to investigate the sound, its 'hearing' circle is disabled.	I decided to change this as I do not think that the speed of the hunter should be increased or that the hearing should be disabled as the speed increase would make it too hard for the player to escape and the disabling of the hearing circle would make the game less realistic. However, I completed this by making the hunter calculate the shortest path to the player using the A* algorithm when the hunter heard the player.
P6	T	If the hunter sees the player then the hunter should start following the player, the speed of the hunter during the chase should be quicker than the walking speed of the player but slower than the sprint speed of the player.	I did this by making the hunter update the path to the player when the hunter sees the player so that the hunter starts following the player.
P7	T	Once the hunter starts chasing the player then the chase sequence needs to be activated which puts a red overlay on the players screen and an exclamation mark above the player's head. Furthermore, the hearing circle will also be disabled in this sequence.	I have decided to change this criterion as I believe that the overlay is unnecessary and adds nothing to the game and the hearing circle shouldn't be disabled as this would make the game less realistic. However, the chase sequence needs to be enabled so that the hunter can follow the player. I did this by having a chase attribute for the hunter so that the path can be recalculated when the hunter is chasing the player.
P8	T	While the player is moving, there will be an area in which the sound can be 'heard' by the hunter, this will be larger while the player is using the sprint button	I changed this success criterion as in the pygame library it is not possible to create a circle as a rect, therefore, the area will just have to be some sort of square
P9	T	The hunter will also have an area around them in which they can hear the player, this circle changes depending on the distance and number of walls between the player and the hunter	I just changed the wording of this success criteria for the same reason as above as it would not be possible to create a circle as a rect so the area around the hunter would have to be a square
P10	T	If the hunter and the player's squares overlap, then depending on the amount	I have also just changed the wording of this success criterion for the same reasons above

		of overlap the hunter will have a different chance on whether they can hear the player or not.	
P11	N/A	If the player is out of the hunters FOV during the chase sequence, then the hunter will follow a random path at a lower movement speed in order to search for the player before reverting to the unalerted version after a few seconds	I have decided to change this as changing the movement speed of the hunter midway through the game causes problems for the hunter so the hunter should just stay at the same speed throughout the whole game. Otherwise, this has been achieved by making the hunter calculate a path to the player as long as the player is within the hunter's field of view and when the player is out of it then the hunter reverts back to random pathfinding
P12	T	If the hunter catches the player then the game will be ended, showing the end screen (O6) and the user will be returned to the start screen.	I have decided to change this as there wasn't enough time to add an end screen to the game and it was not necessary for the experience. However, I have still implemented this by making the game end once the player and hunter have collided with each other.
P13	T7, T8, T9, (S10, S11)	When the player presses the use button ('E') while next to a hiding spot then the players character will be non existent to the hunter while they are still hiding	I did this by giving the player an attribute called hiding and then whenever the method for the player hiding is invoked then that attribute is set to true which is used in various methods to disable different things, for example, the player is not displayed on the screen anymore and also all of the movement controls are disabled so that the player cannot move anymore. This attribute can be accessed via a method from the player so that the hunter can also check if the player is hiding and if the player is hiding then there are places within certain methods such as sight and sound which are disabled for the hunter as well.
P14	N/A	If the player is within the hunter's FOV and player tries to hide using the hiding spot the hunter will still be chasing the player and will catch them if the player is still in that same hiding spot	I have decided to remove this for the same reason I removed the time limit for the player sprint because the map is too small for this feature to be fair on the player since it would make it a lot harder for the player to escape from the hunter.
P15	T7, T8, T9 (S12, S13)	When the player presses the use button 'E' while next to a lever then it will change the light above the lever from red to green and update the objective by one.	I did this by having a method within the screen class which then invokes a method which returns the state of the lever (whether it is activated or deactivated) and as soon as the state of the lever is changed, then the colour is changed through a decision in the method which displays all of the objects.

Name: Muqtasid Zayyan Dar

Project title: Manhunt

P16	N/A	When the player activates the lever then the hunter is alerted to that location after a short amount of time of a few seconds or less.	I decided to remove this as the map in the game is too small for the hunter to be alerted every time the player activates the lever since it would make the game unfair with the hunter going to that exact place without the player even knowing where the hunter is.
P17	T	When the player completes the objective (activating all the levers/switches) then the exit door opens.	I decided to change this as I believe that it would make it more interesting for the player if they had to figure out for themselves when the objective had been complete, furthermore, I decided not to code any unnecessary things such as screens telling the player things which they can already figure out themselves.
P18	T	If the player moves through the escape door then a screen should be displayed saying 'You have escaped' and it also shows how long it took them to complete the game just under the text.	I decided to change this as there was not enough time to add a winning screen to the game and I also believe it was not a necessity for the game. So now the game just ends once the player escapes.
P19	N/A	When the start button is pressed, it takes the user to a different screen which shows different levels of difficulty which will be easy (3 levers); normal (5 levers); hard (7 levers); insane (9 levers) and nightmare (13 levers) - depending on how big the level will be, some of the difficulties may not be included	I had to remove this due to the time constraints since I would not be able to code all of these different levels in the amount of time I had
P20	T	The FOV of the hunter will be a number of projectiles which originate from the hunter and travel in the direction that the hunter is moving, the FOV should not go through walls and only be able to see up till that wall	I decided to change this success criteria as I believe that the projectiles would be more effective and realistic as a FOV and it would be easier to use since I could just change the direction of the projectiles each time the hunter moves in a different direction.
P21	T38 & S1	The FOV of the player will be anything that isn't behind a wall so if a player is standing in the middle of a long hall then they should be able to see to either end of the hall but not beyond the walls next	I created a method which does the field of view for the player, first it uses the map parameter passed into the method in order to get all of the map objects and then it goes through each of the objects and sets their visibility to false so that the object is not visible. Next the method will check if the list attribute holding the sightProjectiles is empty, if it is then it invokes another method which creates 8

Name: Muqtasid Zayyan Dar

Project title: Manhunt

		to them or around the corners at the end of the hall.	sightProjectiles which are all given different x and y speeds so that they will fire in different directions around the player when launched. After they have been created, each projectile is checked to see if it has been launched once or whether they have collided with an object and if either of those is true then the projectile is launched from the player coordinates using the launchSightProjectile method and the list of collided objects which are returned are set as the variable collided which is then appended to the list attribute of the player collided which holds all of the objects that each of the projectiles have collided into. Next, a nested for loop is used to go through each of the objects within the collided attribute and the visible attribute for all of those objects is set to true. Finally the collided attribute is reset to an empty list so that it can check again.
P22	T	The hunter will only be able to catch the player if the hunter is in extremely close proximity to the player	I did this by checking when the player's hitbox collided with the hunter's hitbox which would mean that the player has been caught since the player is close enough to the hunter
P23	T	When the player presses shift while moving then the player should move at a faster speed than when they are just walking. This speed should be faster than the hunters sprint speed	I changed this success criteria to make it less specific as depending on the hunters sprint speed, the player's sprint speed would have to be a specific factor of the normal movement speed in order to be faster than the hunter. I accomplished this by creating an attribute for the player class which is the factor by which the speed of the player is increased
P24	N/A	As the player presses shift (while moving) their stamina bar at the bottom right of the screen will decrease at the same rate as the value for stamina the player has left.	I decided to remove the limit for the player sprint as I believe that the map is too small for there to be a limit to the player's sprint speed
P25	N/A	The players stamina will only replenish if the player is not sprinting (pressing the shift button) and when the stamina value is being replenished then the stamina bar will refill at the same rate.	I decided to remove this for the same reason above, because if there is no limitation to sprinting then there is no need for regeneration of the sprint
P26	T	The level itself will have a simplistic maze design, resembling the inside of some	I did this by creating a grid design for the map with long corridors and different rooms. I made sure that there were no dead ends within the level by making all the paths link together but due to the map having to look like the inside of a building,

Name: Muqtasid Zayyan Dar

Project title: Manhunt

		sort of building. However, there will be no dead ends within the level.	there are some possible dead ends, however, I made sure hiding spaces were available for the player in those places
O1	T	The player's sprite moves upwards if the player presses 'W'	I did this by changing the y attribute for the player and then redisplayed the player using a method with the new coordinates for the player
O2	T	The player's sprite moves downwards if the player presses 'S'	I did this by changing the y attribute for the player and then redisplayed the player using a method with the new coordinates for the player
O3	T	The player's sprite moves left if the player presses 'A'	I did this by changing the x attribute for the player and then redisplayed the player using a method with the new coordinates for the player
O4	T	The player's sprite moves right if the player presses 'D'	I did this by changing the x attribute for the player and then redisplayed the player using a method with the new coordinates for the player
O5	T	If the player tries to move in a direction where an object is directly in front of them, the player's sprite should not go through it	I did this by using a method in the player class which has been inherited from the sprite class. This method takes the parameter map which should be a map object and then first runs the method getWalls on it to get all of the walls in the map as a list as these are the objects the player cannot walk through – this is assigned to a variable called walls, there is also a constant within the method which is given the identifier collisionBounce and determines how many pixels back the player will move when it collides with something. Next the method checks through each wall and the uses a method to get the rect object of the wall, then the method invokes a pygame method on the player's hitbox attribute, which is another rect object, and this method returns true or false based on whether the hitbox has collided with that wall. If it is true then more checks are done using the player's directional attributes to check which direction the player is moving, depending on this, one of the player's coordinate attribute is changed so that they bounce back from the wall.
O6	N/A	If the player gets caught by the hunter then a 'You Have Been Caught' screen will be displayed and the player will be returned to the main menu.	I have removed this as I believe that this screen is unnecessary for the game and I did not have enough time to add this into the game

Name: Muqtasid Zayyan Dar

Project title: Manhunt

O7	N/A	If the player presses 'E' while they are next to something that can be used e.g. a hiding space then the player's sprite should act accordingly and text in relation to the act will be shown e.g. 'Hidden'.	I have removed this success criteria as I believe that seeing the lever turn green on the screen or the player disappear from the screen is enough for the player to know that they have done that action
O8	T	When the player presses the 'shift' button while moving in a direction then the player's sprite should appear to be moving quicker on the monitor.	I accomplished this by creating a boolean attribute for the player called sprinting which is set to true if the user is pressing the shift key and in the method that deals with the player movement, it first checks whether the attribute is true, if it is, then it changes the x or y coordinate by the movement speed multiplied by the scale factor attribute
O9	N/A	If the player complete the objective then a message should appear displaying 'You have escaped' or something similar and return the player to the start page	I have removed this as I believe that this screen is unnecessary for the game and I did not have enough time to add this into the game
O10	N/A	Depending on the objective of the player a relevant display in the corner of how close they are to completing the objective should be in the corner of the screen.	I have decided to remove this from the game as it is not necessary for the player and I believe that it would not affect gameplay for the user if removed.
O11	T	While in the game, the player should have a field of view bigger than the hunter's FOV, however, this is 360 degrees around the player.	As the map is very small, I decided to make the player's FOV unlimited as I believe it would be unrealistic for the player not to be able to see to the end of the map.
O12	N/A	If the player has no more sprinting energy left and the player presses shift then text will be displayed near the top of the screen saying 'Wait for your energy to replenish'	I have removed this as I believe it would be unfair for the player to have to wait to sprint away from the hunter at times as it is such as small map
Ae1	N/A	If the hunter has seen the player and is chasing them, then an alarm effect or something similar should be added on top of the players screen	I have removed this as it would be unnecessary and may get

Name: Muqtasid Zayyan Dar

Project title: Manhunt

Ae2	T	The level itself will be quite basic, with the walls being rectangles which will block the player into a sort of maze which resembles a structure	I achieved this by creating a map which is made up of different types of floor which look like wood, concrete or carpet and then created rooms around the map which contain different types of floors, this makes it resemble the inside of a building
Ae3	T	The player's sprite will look simple and be a circle	I have changed this success criteria as the sprite for the player does not need to look complicated.
Ae4	T	The hunter's sprite will be similar to the players except the hunter will have a different skin to the player	I achieved this by giving the hunter a sprite which is a filled in circle
Ae5	T7, T8, T9	On the players screen places where the player can interact with will look different to the surrounding environment	I achieved this by making the hiding spaces and the levers different colours to the walls around it.

Usability Features:

In my game, I made sure that all of the objects can be easily discerned from each other by making them different colours (S11, S12, S13), for example, I made the walls a different colour to the doors and the doors a different colour to the floor. This makes it easier for the player to know what the different objects in the game. Some of the objects, such as the door and the lever, change colours when they change state, for example, the lever changed colour from red when it has not been activated to green after the player has activated it or how the door changes from its colour when it is closed to white when it is open. Also, the hiding spaces which the player can use are a different colour to all of the other walls and when the player uses the hiding space then their player disappears from the screen so that they know that they are hiding when that happens. Finally, if the user is new to my game and wants to know what the controls are then they are able to easily learn them by clicking on a button in the main menu, this way the user is able to quickly get a grasp of the controls. However, new players may not know what the different colours for the different objects on the map is so this could be improved on.

Maintenance

I have written my code using object oriented programming in order to ensure maximum efficiency when it comes to maintaining it in the future I have done this by using:

- Classes:
 - I separated major parts of my program into different classes, for example, the hunter and player are 2 different classes as they are both major parts of my game. This helps with maintenance as it means that if I need to fix any bugs in the program or I want to add anything then it can be a lot easier.
 - Firstly, if there are any problems with my game that need fixing, then it is easier to track where that problem comes from as I can narrow it down to specific classes or methods depending on what the problem is. Furthermore, when it comes to fixing the problem the change only needs to be done in one place somewhere within the class in order for the effect to take place everywhere in the program. This means I can save time as I will not have to edit the same piece of code in multiple places.
 - If I want to add anything to the program such as a new screen, it is a lot easier since I can just create another object (such as a new screen) from that class, this means that I can be more efficient since I will not have to waste time re-coding the same thing multiple times.
- Inheritance:
 - I used inheritance when multiple classes had similar methods or attributes which could then be used in a super class and then inherited by those multiple subclasses. An example of this is the object super class since this had the common methods and attributes for all of the different objects which were going to appear in the map which were: floors, doors, levers, hiding spaces and walls.
 - This can help when maintaining my program as it means when dealing with the same change to multiple classes, then I can just change that thing in the super class and that change will also take effect in all of the sub classes. This makes it a lot more efficient since I will not have to code the same thing multiple times in separate classes.
- Encapsulation:
 - I used this in all of my classes in order to protect the data held by the classes so that they cannot be changed directly as this could cause an accidental error. I did this by making some methods and most attributes private so that they can only be accessed from within the class via get or set methods.
 - This will make it easier to maintain the code in the future as it means that it will be easier to find errors in the code which are to do with data since it will be easier to find where I have called a method for that class and what I have passed it compared to actually directly changing the attribute.

- This also means that when I need to add more features to my program in the future, I will be able to easily understand what level of access there should be to any of the attributes or the methods and, therefore, how I can use the methods in my program.
- Subroutines:
 - I used functions for the main game loops within the main part of my program
 - This makes it easier to maintain as originally, without the functions, there would need to be multiple nested for loops for each game loop that is used which is needed for every screen, however, by using subroutines, it is a lot easier to follow the flow of the program and to understand what each section of code within a subroutine carries out. This means that when it comes to carrying out bug fixes or adding new things, it is easier to figure out what parts of the code I need to edit, furthermore, as it is a subroutine, any changes made to it will take effect wherever it is invoked in the code. Therefore, it would be more efficient as I would not need to make the same changes in multiple places throughout the code.
- Comments:
 - I used comments throughout the whole program and labelled or annotated different parts to describe what they do or what role they play in the program.
 - This will help maintain the code because, when I come to look at the code again in the future in order to improve it, I may forget exactly what the different parts of the program do, therefore, by using comments I will be able to quickly understand what those parts of the program do without having to waste time figuring out what it does.
 - Comments will also help when it comes to logical errors since these are the hardest errors to find as they are to do with the logic of the code rather than the syntax and by using comments I will be able to figure out where the flaws in the logic of the program may lie.
- Structure:
 - I separated my code into many different files with names which indicate their contents. Within each file, there are also gaps between different sections of code
 - This will make it easier to maintain in the future as I will easily be able to find the different parts of the code which I want to work on since the files are named based on their contents, for example, the gameScreen, Screen and Button class are all within the screens file as they are all used for that purpose.
- Validation:
 - I used validation throughout my classes within the set methods to make sure the data passed into the method is the correct data type
 - This will help with maintenance in the future because when I need to change or modify parts of my code, accidentally passing in the wrong data type in a method could cause an error in the program which would take time to figure out, but by using

Name: Muqtasid Zayyan Dar

Project title: Manhunt

validation, this saves time since I can print something out if the data type is wrong which will not only protect my program but also help me find out where the error is occurring in the program.

Further Developments

Current issues:

Physics – I would like to smooth out the physics bugs in the game which make the player bug across the map, this is most likely to do with how I have approached the collisions between the player and the wall and the time it takes for the program to register that the player has collided with the wall and the actions it needs to carry out. This delay means that an action that should only be carried out once in a collision can occur multiple times if the collision is registered multiple times while the program is trying to carry out the reaction for the collision

Mode buttons – When I tried to implement this function, it was too buggy and was taking too much time, therefore, I decided it would be better to remove it from the current game as I would be able to focus on the main part of my game

Future improvements:

Map size – I would like to make the map size bigger in the future as I believe it would be more fun since the map size could vary and it would make it harder for the player to be able to win the game which would in turn make it more fun and challenging. I would do this by implementing a scrolling screen which would display a part of the map which the player is currently in, as soon as the player moves off the screen towards one side, then the screen would switch and the player would appear in that section of the map. Although only part of the map would be displayed to the player (the part they are currently in) the rest of the map would still have to be working in the background so that the hunter can still be pathfinding around the map looking for the player.

Random Map – I would also like to add a mode where the map is randomised each time so that the player is not able to learn the layout of the level, I would have to do this by first creating an empty map which would just be full of walls, then a certain number of rooms can be loaded in with different floors around the map, these rooms would have to be at least 2 spaces away from each other in every direction (not including the border) so that the player has enough space to move around after the paths are added in the next part. After this then the paths can be generated around the map which would be anywhere that a room has not been generated in. Then finally, the hiding spaces and the levers can be added to the map in any place where there is a wall, however, the only walls where the hiding spaces and the levers can be added in are the walls which are not corner walls, as these walls are not accessible by the player and it would not make sense if the player was able to interact with something inside a wall.

Modes – I would like to add multiple modes to the game where there could be a harder hunter or more levers depending on the mode. I would do this by adding some more functions for the hunter such as being able to deactivate levers, or check hiding spaces randomly within the hunter class. Then there would be a new attribute in the hunter class and based on the mode the player selects, the attribute would be a different value and would enable or disable some of the methods in the class, therefore, altering the behaviour of the hunter. Also, For the different amount of

Name: Muqtasid Zayyan Dar

Project title: Manhunt

levers in the map, I would add another attribute in the map class which functions similarly to the hunter's new attribute, however, this one just changes the number of levers which will be added to the map depending on its value which is determined by the mode.

Hunter hearing – I would like to make the hunter hearing in a less heuristic way so that it can be more realistic as currently, it is based mostly on calculation. I would do this by using the projectiles class which I have made but in a different way, projectiles would be launched in all directions around the hunter and these projectiles would travel until they hit the border walls or they hit a number of walls (which would be about 2 or 3). With every wall the projectiles hit, they move slower to create a dampening effect as sound is dampened after it hits a wall. If any of these projectiles hit the player then based on a set chance, the hunter is able to hear the player, this chance is increased while the player is sprinting.

Limitations

- Graphics – As I wanted to focus more on the complexity of the program and the main problem itself, I had to make the game quite simple with only some images being used for things like the hunter and the player with the rest of the map only using different coloured squares to differentiate between objects
- Multiplayer – As there was not enough time for the project, I did not have time to implement multiplayer features where either there could be many players playing together to escape or one playing as a hunter and one playing as a player.
- Map size – Although I wanted to code a bigger map for the game as I did not have enough time for the project, I was not able to implement a scrolling feature where the player moved around the map and the screen followed the player throughout the map, therefore, I had to stick to a map size which was the same size as the screen
- Modes – Initially, I wanted to implement different modes for the game which would vary in difficulty by making the hunter harder and adding more levers into the map but when I started coding the game, I realised I did not have enough time for this so I had to remove this part of the game from the requirements
- Screens – Although I intended to add a full HUD, game mode screen and make the screen a bit more complex, due to time constraints, I decided that it would be better to leave these parts of the game out so that I could focus on the main complexity for the game.
- 2D – This links to graphics but since making the game 3D would increase the complexity by a lot I was not able to create the game in 3D since it would take too long to program and would not add that much in terms of the features of the program.
- Player FOV – The player FOV that is currently being used works well, however, as python is not a very fast language and the way I have coded it is not the most effective, using too many projectiles makes the game run slow so I was not able to add this in.
- Hunter hearing – Projectiles would make the sound aspect of the hunter hearing more realistic by creating projectiles which originate from the hunter in all directions and if it hits a wall then it may be slowed down by a certain amount (to have a sound dampening effect) and if it hits the player then the hunter would be able to hear the player, however, due to time constraints and the complexity to do this, I had to use a more heuristic approach

Development Style:

I used rapid application development (RAD) in order to code my program. This form of software development helped as the requirements for the project which I had at the start were somewhat unclear and so with each iteration I was able to refine the requirements of my project. Furthermore, as there was continuous feedback from stakeholders for every version, this meant that I was able to suit my game to their requirements. However, since the stakeholders need to give feedback on the iterations very often, it was quite hard to do as not all of my stakeholders were easy to contact, therefore, it was not as efficient as it could have been. As RAD does not have a set schedule for each stage of the development, some parts took longer than was expected such as the development as the requirements for the game changed quite a lot.

The language I used for my program was Python as one of the main requirements of my program was that it should be easily accessible by anyone wherever they want to play it and it can be used on almost any device. However, Python is also a language that I understand very well and by choosing this language it meant that I was able to use the pygame library, this library helped with displaying things on the screen, detecting keyboard inputs and somewhat collisions between objects. This meant that I was able to focus more on the main aspect of my program, such as the hunter, and save time as I did not need to spend as much time thinking about how to display things on the screen or detect what keys the user was pressing. Moreover, the pygame library

During my development, I used a version control system called github in order to keep track of all of the different changes which I made to my program. This was very helpful as it meant I could do things like revert to an older version in case I made a mistake and needed to go back to a specific version to start again. The version control system also allowed me to create branches for my program which meant that I was able to separate code that was being worked on from the tested and stable code. This meant I could change the code as much as I wanted for testing purposes and ensure that the code currently being worked on is thoroughly tested.

By writing psuedocode before creating my main program, I was able to think about the logic and the structure of the program before hand which meant that when it came to coding the solution itself, I already had a good idea of how I was going to do it which saved time.

The decomposition chart for my program helped my figure out what the pieces of the puzzle were for my program with each piece being some function or procedure which would then contribute to the rest of the program.

Code Listing

File: main

```
import pygame
import screens
import maps
import player
import colours
import mapObjects
import hunter

#Initialising the pygame module
pygame.init()

#Setting a title
pygame.display.set_caption("Manhunt")

#Setting the icon
gameIcon = pygame.image.load('Manhunt.png')
pygame.display.set_icon(gameIcon)

#Setting FPS
clock = pygame.time.Clock()

#Screens form screen class
mainMenu = screens.Screen(colours.lightGrey)
settings = screens.Screen(colours.lightGrey)
```

```
mode = screens.Screen(colours.lightGrey)
game = screens.GameScreen(colours.lightGrey)
win = screens.Screen(colours.lightGrey)

#Variables for mainMenu
mainMenu_texts = [ 'Manhunt' ]
mainMenu_textSizes = [160]
mainMenu_textColours = [colours.black]
mainMenu_textFonts = [None]
mainMenu_textCoords = [(230, 80)]
mainMenu_images = []
mainMenu_imagesCoords = []
mainMenu_imagescales = []
mainMenu.renderMTTexts(mainMenu_texts, mainMenu_textSizes, mainMenu_textColours, mainMenu_textFonts,
mainMenu_textCoords)
mainMenu.createButton('start','Start.png', 340, 175, 1)
mainMenu.createButton('options', 'settings.png', 5, 585, 0.1)
mainMenu.addImages(mainMenu_images, mainMenu_imagesCoords, mainMenu_imagescales)

#Variables for settings class
settings_texts = [ 'Settings', 'W: Move Forward', 'A: Move Left', 'S: Move Backward', 'D: Move Right',
'Shift: Sprint', 'E: Interact' ]
settings_textSizes = [160, 50, 50, 50, 50, 50, 50, 50]
settings_textColours = [colours.black, colours.black, colours.black, colours.black, colours.black,
colours.black, colours.black]
settings_textFonts = [None, None, None, None, None, None, None, None]
settings_textCoords = [(230, 80), (300,250), (300,310), (300,370), (300,430), (300,490), (300, 550)]
settings_images = []
settings_imagesCoords = []
settings_imageScales = []
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
settings.renderMTexts(settings_texts, settings_textSizes, settings_textColours, settings_textFonts,
settings_textCoords)
settings.createButton('return','return.png', 10, 10, 0.1)
settings.addImages(settings_images, settings_imagesCoords, settings_imageScales)

#Variables for GameScreen class
game_texts = []
game_textSizes = []
game_textColours = []
game_textFonts = []
game_textCoords = []
game_images = []
game_imageCoords = []
game_imageScales = []
game.renderMTexts(game_texts, game_textSizes, game_textColours, game_textFonts, game_textCoords)
game.addImages(game_images, game_imageCoords, game_imageScales)

def mainMenuScreen(mainMenu, settings, clock):
    running = True
    while running:
        clock.tick(60)
        mainMenu.displayScreen()
        if mainMenu.searchButton('start').clickCheck(mainMenu.screen) == True:
            running = gameScreen(clock)
            print('Start')
        if mainMenu.searchButton('options').clickCheck(mainMenu.screen) == True:
            running = settingsScreen(settings, clock)
            print('settings')
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
```

```
        running = False
        pygame.display.update()

def settingsScreen(settings, clock):
    running = True
    while running:
        clock.tick(60)
        settings.displayScreen()
        if settings.searchButton('return').clickCheck(settings.screen) == True:
            return True
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False

        pygame.display.update()

#Creating the map for the game
walls = {'empty':mapObjects.empty, 'hidingSpace':mapObjects.hidingSpace, 'lever':mapObjects.lever}
floors = {'wood':mapObjects.wood, 'concrete':mapObjects.concrete, 'carpet':mapObjects.carpet}
doorCoord = mapObjects.doors
map = maps.Map(walls, floors, doorCoord, 20, 30)
map.createMap()
mapList = map.getMap()

playerX = 3
playerY = 6
playerOne = player.Player(playerX * 32, playerY * 32, 'BlueCircle.png', 1, 2, 2, map)
playerOne.setMaxLevers(map)
```

Name: Muqtasid Zayyan Dar

Project title: Manhunt

```
hunterX = 15
hunterY = 10
hunterOne = hunter.Hunter(hunterX * 32, hunterY * 32, 'RedCircle.png', 1, 2, 2)

def gameScreen(clock):
    running = True
    done = 0
    while running:
        clock.tick(120)
        game.displayGameScreen(map.getMap())
        playerWin = playerOne.ready(map, game, hunterOne)
        hunterWin = hunterOne.ready(game.getScreen(), map, playerOne)
        if playerWin:
            return False
            running = winScreen(clock)
        if hunterWin:
            return False
            running = winScreen(clock)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False
            playerOne.interactCheck(event, map)

        pygame.display.update()

mainMenuScreen(mainMenu, settings, clock)
```

File: colours

```
black = (0, 0, 0)
red = (155, 0, 0)
white = (255, 255, 255)
lightGrey = (150, 150, 150)
green = (0, 255, 0)
blue = (0, 0, 255)
navy = (0, 0, 128)
lightBrown = (199, 150, 98)
darkBrown = (79, 60, 39)
darkGrey = (100, 100, 100)
```

File: hunter

```
import pygame
import sprite
import math
import physics
import random
import objects
pygame.init()

#class AStar():

class Hunter(sprite.Sprite):
    def __init__(self, x, y, img, scale, speed, sprintMultiplier):
        super().__init__(x, y, img, scale, speed, sprintMultiplier)
        self.soundProj = None
```

```
self.soundProjSpeed = 16
self.heard = False

self.visible = False

self.sightProj = []
self.seen = False

self.firstMove = False

self.path = None
self.pathIndex = 0

self.chasing = False

def displayHunter(self, screen):
    self.displaySprite(screen)

def pythagoras(self, x, y):
    xSqr = x**2
    ySqr = y**2
    dist = math.sqrt(xSqr + ySqr)
    return dist

#This method returns what values the xSpeed and ySpeed of the sound projectile should be
def checkDirection(self, xDist, yDist):
    xSign = 0
    ySign = 0
    if xDist < 0:
        xSign = -1
    elif xDist > 0:
```

```
xSign = 1
if yDist < 0:
    ySign = -1
elif yDist > 0:
    ySign = 1
return xSign, ySign

#This method takes in the absolute values for the x and y distances and then returns
#the angle at which the projectile needs to be launched
def angleCalcR(self, xDist, yDist):
    o = abs(yDist)
    a = abs(xDist)
    value = o/a
    angle = math.atan(value)
    return angle

#This method takes the angle at which the projectile needs to be launched and
#uses trigonometry in order to calculate what the x and y component need to be
#in order for the projectile to move at a certain speed in that direction
def speedCalcR(self, angle):
    xSpeed = 0
    ySpeed = 0
    projSpeed = self.soundProjSpeed
    cos = math.cos(angle)
    sin = math.sin(angle)
    xSpeed = projSpeed * cos
    ySpeed = projSpeed * sin
    return xSpeed, ySpeed

#This method checks the number of walls between the player and the hunter
def checkWalls(self, player, screen, map):
```

```
xSpeed = 0
ySpeed = 0
#Checks if the sound projectile object has been created
if self.soundProj == None:
    self.createSoundProjectile()
if self.soundProj.getLaunched() == False or self.soundProj.getCollided() == True:
    self.soundProj.setCollided(False)
#Sets the attribute for the sound projectile as a variable
soundProj = self.soundProj
#Gets the player coordinates on the map
pCoords = player.getCoords()
#Calculates the distance between the player and the hunter in each axial direction
coordDist = self.coordinateDistance(pCoords)
xDist = coordDist[0]
yDist = coordDist[1]
direction = self.checkDirection(xDist, yDist)
if xDist != 0 and yDist != 0:
    angle = self.angleCalcR(xDist, yDist)
    speeds = self.speedCalcR(angle)
    xSpeed = speeds[0] * direction[0]
    ySpeed = speeds[1] * direction[1]
elif xDist == 0:
    xSpeed = 0
    ySpeed = self.soundProjSpeed * direction[1]
elif yDist == 0:
    ySpeed = 0
    xSpeed = self.soundProjSpeed * direction[0]
soundProj.setXSpeed(xSpeed)
soundProj.setYSpeed(ySpeed)
walls = soundProj.launchSoundProjectile(screen, self.getHitbox().center, map, player)
return walls
```

```
else:
    return

#This method calculates the distance in each axis
def coordinateDistance(self, coords):
    hCoords = self.getCoords()
    startX = coords[0]
    endX = hCoords[0]
    startY = coords[1]
    endY = hCoords[1]
    xDist = startX - endX
    yDist = startY - endY
    return xDist, yDist

def calcDistance(self, player):
    playerCoords = player.getCoords()
    xDist, yDist = self.coordinateDistance(playerCoords)
    dist = self.pythagoras(xDist, yDist)
    return dist

#This method calculates whether the hunter should be able to hear the player or not
#and returns a boolean value based on that
def sound(self, player, screen, map):
    maxSound = 100
    soundLevel = player.getSound()
    dist = self.calcDistance(player)
    if dist > 600:
        return False
    if isinstance(soundLevel, float) and player.getMovement():
        maxSound = soundLevel * maxSound
    wallNum = self.checkWalls(player, screen, map)
```

```
maxSound -= wallNum
if maxSound <= 50:
    self.setHeard(False)
elif maxSound <= 75:
    chance = random.randint(51, 75)
    if chance == maxSound:
        self.setHeard(True)
    else:
        self.setHeard(False)
elif maxSound <= 85:
    chance = random.randint(76, 85)
    if chance == maxSound:
        self.setHeard(True)
    else:
        self.setHeard(False)
else:
    self.setHeard(True)

def getHeard(self):
    return self.heard

def setHeard(self, value):
    if value != True and value != False:
        print('INCORRECT VALUE FOR HEARD')
    self.heard = value

def setVisible(self, value):
    if value != True and value != False:
        print('NOT BOOLEAN VALUE FOR H VISIBILITY')
```

```
    self.visible = value

def createSoundProjectile(self):
    self.soundProj = physics.SoundProjectile(self.getHitbox().center, 0, 0, 1)

def setSeen(self, value):
    if value != True and value != False:
        print('VALUE FOR SEEN IS NOT BOOLEAN')
    self.seen = value

def getSeen(self):
    return self.seen

def createSightProjectile(self):
    for x in range(0, 3):
        proj = physics.HunterSightProjectile(self.getHitbox().center, 0, 0, 32)
        self.sightProj.append(proj)

def fov(self, screen, map, player):
    allWalls = map.getWalls()
    allWalls.append(player)
    projSpeed = 20
    if len(self.sightProj) == 0:
        self.createSightProjectile()

    self.setProjDirection(projSpeed)
    collided = False
    for proj in self.sightProj:
        if proj.getCollided() == True or proj.getLaunched() == False:
            proj.setCollided(False)
```

```
        check = proj.launchSightProjectile(screen, self.getHitbox().center, allWalls)
        if check == True:
            collided = True
        self.setSeen(collided)

def setProjDirection(self, projSpeed):
    if self.getBackward():
        #LEFT
        self.sightProj[0].setXSpeed(projSpeed * -1)
        self.sightProj[0].setYSpeed(projSpeed)
        #MIDDLE
        self.sightProj[1].setXSpeed(0)
        self.sightProj[1].setYSpeed(projSpeed)
        #RIGHT
        self.sightProj[2].setXSpeed(projSpeed)
        self.sightProj[2].setYSpeed(projSpeed)

    elif self.getRight():
        #TOP
        self.sightProj[0].setXSpeed(projSpeed)
        self.sightProj[0].setYSpeed(projSpeed * -1)
        #MIDDLE
        self.sightProj[1].setXSpeed(projSpeed)
        self.sightProj[1].setYSpeed(0)
        #BOTTOM
        self.sightProj[2].setXSpeed(projSpeed)
        self.sightProj[2].setYSpeed(projSpeed)

    elif self.getLeft():
        #TOP
        self.sightProj[0].setXSpeed(projSpeed * -1)
```

```
        self.sightProj[0].setYSpeed(projSpeed * -1)
        #MIDDLE
        self.sightProj[1].setXSpeed(projSpeed * -1)
        self.sightProj[1].setYSpeed(0)
        #BOTTOM
        self.sightProj[2].setXSpeed(projSpeed * -1)
        self.sightProj[2].setYSpeed(projSpeed)

    else:
        #LEFT
        self.sightProj[0].setXSpeed(projSpeed * -1)
        self.sightProj[0].setYSpeed(projSpeed * -1)
        #MIDDLE
        self.sightProj[1].setXSpeed(0)
        self.sightProj[1].setYSpeed(projSpeed * -1)
        #RIGHT
        self.sightProj[2].setXSpeed(projSpeed)
        self.sightProj[2].setYSpeed(projSpeed * -1)

def randomMove(self):
    x = random.randint(1, 4)
    if x == 1:
        self.moveForward()
    if x == 2:
        self.moveBackward()
    if x == 3:
        self.moveRight()
    if x == 4:
        self.moveLeft()

#This method runs a method on all of the objects in the map to calculate the hCost
```

```
#for each object from that object to the end goal coordinates
def calcHCost(self, map, endCoords):
    allFloors = map.getFloors()
    for floor in allFloors:
        floor.reset()
        floor.calcHCost(endCoords)

#This is the code for the A* algorithm for the hunter to use
def aStar(self, endCoords, map):
    print('calculating')
    #This variable is for whether the point has been found
    found = False
    #These are the open and closed lists for the nodes
    open = []
    closed = []
    #The start node is wherever the hunter is
    startCoords = self.getMapCoords()
    startNode = map.getObject(startCoords)
    #This method is run in order to determine the H cost for every node to the endpoint
    self.calcHCost(map, endCoords)
    #As the start node is no distance from itself, the gCost for it is 0
    startNode.setGFCost(0)
    #The start node is added to the open list as its the first one to be considered
    open.append(startNode)

    #This loop runs while the endpoint has not been found
    while found == False:
        #Current is the lowest f cost in the open list,
        #This must be a high number so that it does not interfere with actual f costs
        lowestF = 100000000
        current = None
```

```
#This loops through each element in the open list
for x in range(len(open)):
    #Checks if the f cost of the current node is greater than any other nodes in the open
    #list and if it is then the node with the lower f cost is set as the current
    if open[x].getFCost() < lowestF:
        current = open[x]
        lowestF = current.getFCost()
        index = x
    #After the node with the lowest fCost is found then the node is added to the closed list
    closed.append(current)
    #and deleted from the open list
    del open[index]

    #This checks if the current node is the node which we are looking for
    #If it is then the loop will be broken
    if current.getCoords() == endCoords:
        found = True

    #This goes through each of the neighbours of the current node
    for node in self.getNeighbours(current, map):
        #This checks if the node is an object of type floor and that it is not already in
closed
        if isinstance(node, objects.Floor) and self.checkClosed(closed, node):
            #This sets the last node for the neighbour node as the current
            node.setLast(current)
            #This sets the g and f cost of the node now based on the last node
            node.setGFCost()
            #This checks if the node
            open.append(node)
print('calculated')
```

```
return current

#This method is a recursive algorithm which goes through
#each node in the path and gets the previous node by invoking the same function
def findPath(self, endNode):
    #Base case - if g cost is 0 it must be the place where we began
    if endNode.getGCost() == 0:
        return [endNode]
    endNode.path = True
    list = self.findPath(endNode.getLast())
    list.append(endNode)
    return list

#This method takes a node on the map and returns the neighbours of the node
def getNeighbours(self, node, map):
    coords = node.getCoords()
    cOne = (coords[0] - 1, coords[1])
    cTwo = (coords[0] + 1, coords[1])
    cThree = (coords[0], coords[1] - 1)
    cFour = (coords[0], coords[1] + 1)
    nOne = map.getObject(cOne)
    nTwo = map.getObject(cTwo)
    nThree = map.getObject(cThree)
    nFour = map.getObject(cFour)
    neighbours = [nOne, nTwo, nThree, nFour]
    return neighbours

#This method checks whether a node is already in the closed list
def checkClosed(self, closed, current):
    for node in closed:
```

```
if node == current:  
    return False  
return True  
  
#This method checks the hunter's position relative to a node  
#and based on that moves in a specific direction  
def traverse(self, node, map):  
    hCoords = self.getHitbox().center  
    gCoords = node.getRect().center  
    xH, yH = hCoords  
    xG, yG = gCoords  
    xDir = xH - xG  
    yDir = yH - yG  
    if xDir > 0:  
        self.moveLeft()  
        self.setCoords()  
        self.checkCollision(map)  
    if xDir < 0:  
        self.moveRight()  
        self.setCoords()  
        self.checkCollision(map)  
    if yDir > 0:  
        self.moveForward()  
        self.setCoords()  
        self.checkCollision(map)  
    if yDir < 0:  
        self.moveBackward()  
        self.setCoords()  
        self.checkCollision(map)
```

```
#This is the overarching method for the pathfinding algorithm which includes the movement and the calculations
def pathfind(self, endCoords, map):
    endNode = self.aStar(endCoords, map)
    self.path = self.findPath(endNode)
    self.pathIndex = 0

#This method checks whether the hunter has a path to follow or not
#and based on this it either makes the hunter follow that path or
#reset the variables for another path
def followPath(self, map):
    if self.pathIndex < len(self.path):
        node = self.path[self.pathIndex]
        self.traverse(node, map)
        nodeCoords = node.getRect().center
        if nodeCoords == self.getCoords():
            self.pathIndex += 1

    else:
        self.path = None

#This method checks whether the hunter needs to create a new path or not and
#it also checks if the hunter has either seen or heard the player.
def checkPath(self, map, player):
    hiding = player.getHiding()
    if self.path != None:
        if (self.heard == True or self.seen == True) and hiding == False and self.getChasing() == False:
            endCoords = player.getMapCoords()
```

```
        self.pathfind(endCoords, map)
        self.setChasing(True)
    else:
        self.setChasing(False)
        self.randomPath(map)
    self.followPath(map)

#This method generates a random path for the hunter to follow
def randomPath(self, map):
    floors = map.getFloors()
    index = random.randint(0, (len(floors) - 1))
    floor = floors[index]
    endCoords = floor.getCoords()
    self.pathfind(endCoords, map)

#This method checks if the hunter has won the game
def checkWin(self, player):
    pRect = player.getHitbox()
    hRect = self.getHitbox()
    if hRect.colliderect(pRect) and player.getHiding() == False:
        return True

def getChasing(self):
    return self.chasing

def setChasing(self, value):
    self.chasing = value

def ready(self, screen, map, player):
    self.setCoords()
```

```
#if self.visible == True:  
    self.displayHunter(screen)  
    self.sound(player, screen, map)  
    self.fov(screen, map, player)  
    self.checkPath(map, player)  
    win = self.checkWin(player)  
    if win == True:  
        return True
```

File: mapObjects

```
empty = [(24, 11), (25, 11), (26, 11), (27, 11), (28, 11), (24, 12), (24, 13), (24, 14),  
         (24, 15), (24, 16), (16, 11), (17, 11), (18, 11), (19, 11), (20, 11), (21, 11),  
         (16, 12), (16, 13), (16, 14), (16, 15), (16, 16), (16, 17), (16, 18), (13, 16),  
         (13, 17), (13, 18), (13, 11), (13, 12), (13, 13), (6, 11), (7, 11), (8, 11), (9, 11),  
         (10, 11), (10, 13), (10, 14), (3, 15), (4, 15), (5, 15), (6, 15),  
         (7, 15), (8, 15), (9, 15), (10, 15), (3, 12), (3, 13), (3, 14), (1, 11), (2, 11), (3, 11),  
         (1, 8), (2, 8), (3, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (12, 8), (13, 8),  
         (6, 5), (6, 6), (6, 7), (3, 4), (4, 4), (5, 4), (6, 4), (13, 3), (13, 4), (13, 5), (13, 6),  
         (13, 7), (16, 8), (17, 8), (18, 8), (19, 8), (20, 8), (21, 8), (22, 8), (23, 8), (24, 8),  
         (25, 8),  
         (26, 8), (27, 8), (28, 8), (17, 5), (18, 5), (19, 5), (20, 5), (21, 5), (22, 5), (23, 5),  
         (24, 5),  
         (25, 5), (26, 5), (16, 1), (16, 2), (16, 3), (16, 4), (16, 5), (22, 1), (22, 2), (21, 18),  
         (21, 16),  
         (21, 13), (8, 4), (12, 4), (19, 2), (25, 1)]  
]  
  
hidingSpace = [(29, 17), (21, 12), (10, 12), (7, 19), (11, 4), (0, 1), (25, 2)]  
  
lever = [(29, 13), (21, 17), (6, 12), (0, 13), (7, 4), (19, 1), (29, 6)]
```

```
wood = [(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (4, 1), (4, 2),
        (4, 3), (5, 1), (5, 2), (5, 3), (6, 1), (6, 2), (6, 3), (7, 1), (7, 2), (7, 3), (8, 1),
        (8, 2), (8, 3), (9, 1), (9, 2), (9, 3), (10, 1), (10, 2), (10, 3), (11, 1), (11, 2), (11, 3),
        (12, 1), (12, 2), (12, 3), (17, 3), (17, 4), (18, 3), (18, 4), (19, 3), (19, 4), (20, 3), (20,
4),
        (21, 3), (21, 4), (22, 3), (22, 4), (23, 3), (23, 4), (24, 3), (24, 4), (25, 3), (25, 4), (26,
3),
        (26, 4), (27, 3), (27, 4), (28, 3), (28, 4), (16, 6), (16, 7), (17, 6), (17, 7), (18, 6), (18,
7),
        (19, 6), (19, 7), (20, 6), (20, 7), (21, 6), (21, 7), (22, 6), (22, 7), (23, 6), (23, 7), (24,
6),
        (24, 7), (25, 6), (25, 7), (26, 6), (26, 7), (27, 6), (27, 7), (28, 6), (28, 7), (22, 11),
(22, 12),
        (22, 13), (22, 14), (22, 15), (22, 16), (22, 17), (22, 18), (23, 11), (23, 12), (23, 13), (23,
14),
        (23, 15), (23, 16), (23, 17), (23, 18), (11, 11), (11, 12), (11, 13), (11, 14), (11, 15), (12,
11),
        (12, 12), (12, 13), (12, 14), (12, 15), (7, 12), (7, 13), (7, 14), (8, 12), (8, 13), (8, 14),
(9, 12),
        (9, 13), (9, 14), (26, 1), (26, 2), (27, 1), (27, 2), (28, 1), (28, 2), (13, 1), (13, 2), (27,
4), (28, 4),
        (13, 14), (13, 15), (6, 13), (6, 14), (27, 5), (28, 5)
]

concrete = [(1, 9), (1, 10), (2, 9), (2, 10), (3, 9), (3, 10), (4, 9), (4, 10), (5, 9), (5, 10), (6,
9),
            (6, 10), (7, 9), (7, 10), (8, 9), (8, 10), (9, 9), (9, 10), (10, 9), (10, 10), (11, 9),
(11, 10),
            (12, 9), (12, 10), (13, 9), (13, 10), (14, 9), (14, 10), (15, 9), (15, 10), (16, 9), (16,
10), (17, 9),
```

```
(17, 10), (18, 9), (18, 10), (19, 9), (19, 10), (20, 9), (20, 10), (21, 9), (21, 10), (22,  
9), (22, 10),  
        (23, 9), (23, 10), (24, 9), (24, 10), (25, 9), (25, 10), (26, 9), (26, 10), (27, 9), (27,  
10), (28, 9),  
        (28, 10), (14, 11), (14, 12), (14, 13), (14, 14), (14, 15), (14, 16), (14, 17), (14, 18),  
(15, 11), (15, 12),  
        (15, 13), (15, 14), (15, 15), (15, 16), (15, 17), (15, 18), (14, 1), (14, 2), (14, 3),  
(14, 4), (14, 5), (14, 6),  
        (14, 7), (14, 8), (15, 1), (15, 2), (15, 3), (15, 4), (15, 5), (15, 6), (15, 7), (15, 8)  
]  
  
carpet = [(1, 5), (1, 6), (1, 7), (2, 5), (2, 6), (2, 7), (3, 5), (3, 6), (3, 7), (4, 5), (4, 6), (4,  
7), (5, 5),  
        (5, 6), (5, 7), (7, 5), (7, 6), (7, 7), (8, 5), (8, 6), (8, 7), (9, 5), (9, 6), (9, 7), (10,  
5), (10, 6),  
        (10, 7), (11, 5), (11, 6), (11, 7), (12, 5), (12, 6), (12, 7), (17, 1), (17, 2), (18, 1),  
(18, 2), (20, 1),  
        (20, 2), (21, 1), (21, 2), (23, 1), (23, 2), (24, 1), (24, 2), (25, 12), (25, 13), (25, 14),  
(25, 15), (25, 16),  
        (25, 17), (25, 18), (26, 12), (26, 13), (26, 14), (26, 15), (26, 16), (26, 17), (26, 18),  
(27, 12), (27, 13), (27, 14),  
        (27, 15), (27, 16), (27, 17), (27, 18), (28, 12), (28, 13), (28, 14), (28, 15), (28, 16),  
(28, 17), (28, 18), (17, 12),  
        (17, 13), (17, 14), (17, 15), (17, 16), (17, 17), (17, 18), (18, 12), (18, 13), (18, 14),  
(18, 15), (18, 16), (18, 17),  
        (18, 18), (19, 12), (19, 13), (19, 14), (19, 15), (19, 16), (19, 17), (19, 18), (20, 12),  
(20, 13), (20, 14), (20, 15),  
        (20, 16), (20, 17), (20, 18), (4, 16), (4, 17), (4, 18), (5, 16), (5, 17), (5, 18), (6, 16),  
(6, 17), (6, 18), (7, 16),  
        (7, 17), (7, 18), (8, 16), (8, 17), (8, 18), (9, 16), (9, 17), (9, 18), (10, 16), (10, 17),  
(10, 18), (11, 16), (11, 17),
```

```
(11, 18), (12, 16), (12, 17), (12, 18), (4, 11), (4, 12), (4, 13), (4, 14), (5, 11), (5, 12), (5, 13), (5, 14), (1, 12),
(1, 13), (1, 14), (1, 15), (1, 16), (1, 17), (1, 18), (2, 12), (2, 13), (2, 14), (2, 15),
(2, 16), (2, 17), (2, 18), (4, 8),
(5, 8), (1, 4), (2, 4), (21, 14), (21, 15), (24, 17), (24, 18), (9, 4), (10, 4), (3, 16),
(3, 17), (3, 18)
]

doors = [(15, 19), (14, 19)]
```

File: maps

```
import pygame
import objects

pygame.init()

class Map():
    #walls and floors are passed as dictionaries where the key
    #corresponds to the type of each and also includes the coordinates
    #door is passed as just coordinates
    #The coordinates in each of these are not the display coordinates but
    #the grid coordinates
    def __init__(self, walls, floors, doorCoords, mapHeight, mapWidth):
        #doorCoord is the coordinates of the node with the door
        #not the display coordinates
        self.doorCoords = doorCoords
        #This is passed as a dictionary which only contains the coordinates of
        #Each type of wall
        self.walls = walls
        self.borderWalls = []
```

```
#This is also passed as a dictionary with similar things
self.floors = floors
self.map = []
#As there needs to be an outer border of empty walls,
#The real height and width of the map will need to be increased by 2
#self.mapLength = mapLength + 2
self.mapHeight = mapHeight
self.mapWidth = mapWidth

def getMap(self):
    return self.map

#Uses the different methods within the class to create a map
def createMap(self):
    self.createEmptyMap()
    self.addWalls()
    self.addFloors()
    self.addDoor()

#Creates an empty shell for the map with an outer border of walls and empty spaces in the middle
def createEmptyMap(self):
    #Loops for as long as the length of the map is
    for y in range(self.mapHeight):
        #for every loop a new list is appended
        self.map.append([])
        for x in range(self.mapWidth):
            #for every loop a new element is appended to the new list
            if y == 0 or y == self.mapHeight - 1:
                #This if statement checks if it is the first list or the last list
                #and if it is it fills it with wall objects
                wall = objects.Wall((x, y))
```

```
        self.map[y].append(wall)
        self.borderWalls.append((x,y))
    elif x == 0 or x == self.mapWidth - 1:
        #This checks if the element is the first or the last
        #and if it is then it appends a wall object instead
        wall = objects.Wall((x, y))
        self.map[y].append(wall)
        self.borderWalls.append((x,y))
    else:
        #This appends an empty spot in the list
        self.map[y].append(0)

#Adds the inner walls to the map
def addWalls(self):
    #Goes through each of the keys in the dictionary
    for type in self.walls:
        #Uses the key to access the list associated with each key
        #Goes through each element in the list which is the coordinates
        #of that wall
        for coord in self.walls[type]:
            #Creates a wall object
            wall = objects.Wall(coord, type)
            #Switches one of the empty spots (0) to that wall object
            self.map[coord[1]][coord[0]] = wall
    for wall in self.borderWalls:
        self.walls['empty'].append(wall)

#Adds the floors to the map
def addFloors(self):
    #Goes through each type of floor in self.floor dictionary
    for type in self.floors:
```

```
#Uses the key to access coordinates of type of floor
for coord in self.floors[type]:
    #Creates an object of type floor, passing the coordinates and type
    floor = objects.Floor(coord, type)
    #Changes an empty spot on the map to that object
    self.map[coord[1]][coord[0]] = floor

#Procedure that adds the door to the map
def addDoor(self):
    for doorCoord in self.doorCoords:
        door = objects.Door(doorCoord)
        self.map[doorCoord[1]][doorCoord[0]] = door

#Function that returns all of the walls or a type of wall
def getWallCoords(self, category = None):
    allWalls = []
    if category == None:
        for type in self.walls:
            for wall in self.walls[type]:
                allWalls.append(wall)

    return allWalls
else:
    return self.walls[category]

#Function that returns all of the floors or all of a single type of floor
def getFloorCoords(self, category = None):
    allFloorCoords = []
    if category == None:
        for type in self.floors:
            for floor in self.floors[type]:
```

```
        allFloorCoords.append(floor)
    return allFloorCoords
else:
    return self.floors[category]

#Function that returns all of the wall objects
def getWalls(self, category = None):
    allWalls = []
    coords = self.getWallCoords(category)
    for coord in coords:
        wall = self.map[coord[1]][coord[0]]
        allWalls.append(wall)
    return allWalls

#Function that returns all of the floor objects
def getFloors(self, category = None):
    allFloors = []
    coords = self.getFloorCoords(category)
    for coord in coords:
        floor = self.map[coord[1]][coord[0]]
        allFloors.append(floor)
    return allFloors

#Function that returns an object based on the coordinates
def getObject(self, coords):
    return self.map[coords[1]][coords[0]]

#Function that returns all of the walls which the player can interact with (walls which have
levers etc)
def getItemWalls(self):
```

```
HWalls = self.getWalls('hidingSpace')
LWalls = self.getWalls('lever')
interactables = []

for HWall in HWalls:
    interactables.append(HWall)
for LWall in LWalls:
    interactables.append(LWall)

return interactables

#Function that returns the door object based on the coordinates of the door
def getDoors(self):
    doorList = []
    for doorCoord in self.doorCoords:
        door = self.map[doorCoord[1]][doorCoord[0]]
        doorList.append(door)
    return doorList

#This returns all of the objects as 1 complete list
def getAllObjects(self):
    walls = self.getWalls()
    floors = self.getFloors()
    doors = self.getDoors()
    all = []
    for wall in walls:
        all.append(wall)
    for floor in floors:
        all.append(floor)
    for door in doors:
        all.append(door)
```

```
    return all
```

File: objects

```
import pygame
import math
pygame.init()

class Object():
    def __init__(self, Coords):
        #Each object has coordinates on the map
        self.Coords = Coords
        #Creates a rect using the height and width of the object which will be used to display the
object
        self.rect = pygame.Rect(self.Coords[0] * 32, self.Coords[1] * 32, 32, 32)
        #This is the center cooridnates of the object on the screen
        self.center = self.rect.center
        self.visible = False
        self.hCost = 0
        self.fCost = 0
        self.gCost = 0
        self.last = None
        self.path = False

    def reset(self):
        self.hCost = 0
        self.fCost = 0
        self.gCost = 0
        self.last = None
```

```
    self.path = False

def setLast(self, value):
    self.last = value

def getLast(self):
    return self.last

#Function returns the coordinates of the object
def getCoords(self):
    return self.Coords

def getRect(self):
    return self.rect

def getCenter(self):
    return self.center

def getVisible(self):
    return self.visible

def setVisible(self, value):
    if value == True or value == False:
        self.visible = value
    else:
        print('VALUE IS NOT A BOOLEAN')

def pythagoras(self, a, b):
    sqra = a**2
    sqrb = b**2
    sqrc = sqra + sqrb
```

```
c = math.sqrt(sqrc)
c = round(c)
return c

def calcHCost(self, endCoords):
    startCoords = self.getCoords()
    xDist = endCoords[0] - startCoords[0]
    yDist = endCoords[1] - startCoords[1]
    dist = self.pythagoras(xDist, yDist)
    self.hCost = dist

def setFCost(self):
    self.fCost = self.gCost + self.hCost

def setGCost(self, value):
    self.gCost = value

def setGFCost(self, value = None):
    if value != None:
        self.setGCost(0)
    else:
        last = self.last
        self.setGCost(last.getGCost() + 1)
    self.setFCost()

def getGCost(self):
    return self.gCost

def getFCost(self):
    return self.fCost
```

```
def getHCost(self):
    return self.hCost

class Wall(Object):
    def __init__(self, Coords, type = None):
        super().__init__(Coords)
        self.type = type
        #This sets the wall as
        self.item = self.setItem()

    def setItem(self):
        #If type is none that means that there is nothing on the wall
        if self.type == 'empty':
            return None

        #If type is 0 then that means that there is a hidingSpace on the wall
        if self.type == 'hidingSpace':
            return HidingSpace(self.Coords)

        #If type is 1 then that means that there is a lever on the wall
        if self.type == 'lever':
            return Lever(self.Coords)

    def getItem(self):
        return self.item
```

```
class Floor(Object):
    def __init__(self, Coords, type):
        super().__init__(Coords)
        self.type = type
        #Stores the level of sound for each floor object depending on the type of floor
        self.soundLevel = self.setSound()

    #Procedure sets the sound level depending on the type of the floor
    def setSound(self):
        if self.type == 'carpet':
            return 0.3

        if self.type == 'concrete':
            return 0.6

        if self.type == 'wood':
            return 0.9

    #Function returns the sound level of the floor
    def getSoundLevel(self):
        return self.soundLevel

    def getType(self):
        return self.type


class Lever(Object):
    def __init__(self, Coords):
        super().__init__(Coords)
```

```
self.activated = False

#Function checks if lever has been activated
def getActivated(self):
    return self.activated

#These 2 procedures change the self.activated attribute
def activate(self):
    self.activated = True

def deactivate(self):
    self.activated = False

class HidingSpace(Object):
    def __init__(self, Coords):
        super().__init__(Coords)
        self.activationArea = pygame.Rect((self.Coords[0] - 1) * 32, (self.Coords[1]) * 32, 96, 96)

    def inArea(self, playerHitbox):
        check = self.checkCollision(playerHitbox, self.activationArea)
        return check

class Door(Object):
    def __init__(self, Coords):
        super().__init__(Coords)
        self.activated = False
```

```
#Function that returns whether the door has been activated
def checkDoorActivation(self):
    return self.activated

#Checks if the door should be activated depending on the max number of levers and how many have
been activated
def activate(self):
    self.activated = True

def getActivated(self):
    return self.activated

def getColour(self):
    return self.colour
```

File: physics

```
import pygame
import math
import objects
import hunter
import colours
import player
pygame.init()

class Physics():
    def __init__(self):
```

```
pass

def checkCollision(self, objOne, objTwo):
    return objOne.colliderect(objTwo)

def pythagoras(self, numOne, numTwo):
    squared = (numOne ** 2) + (numTwo ** 2)
    squareRoot = math.sqrt(squared)
    return squareRoot

#Parent class for sight projectiles and sound projectiles
class Projectile():
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__()
        #Control the speed of the projectile
        self.xSpeed = xSpeed
        self.ySpeed = ySpeed
        #Origin coordinates of the projectile
        self.coords = coords
        #This is the actual rect for the projectile, created using pygame library
        self.rect = pygame.Rect(coords[0], coords[1], length, length)
        #This is a boolean value which is whether the projectile has collided with a wall or not
        self.collided = False
        #This is a boolean value which is whether the projectile has been launched or not
        self.launched = False

    #This is a get function which returns the value for the launched attribute
    def getLaunched(self):
        return self.launched
```

```
#This is the get function returning the value for the collided attribute
def getCollided(self):
    return self.collided

#This is a function that checks if an object passed has collided with itself
def collideCheck(self, object):
    return self.rect.colliderect(object)

#This is a procedure that sets the value of the collided attribute as a
#value passed to the procedure
def setCollided(self, value):
    if value == True or value == False:
        self.collided = value

#This is the procedure that moves the projectile, screen is for testing purposes
def moveProjectile(self, screen):
    self.rect.x += self.xSpeed
    self.rect.y += self.ySpeed

#This method changes the x speed of the projectile to whatever is passed into the method
def setXSpeed(self, value):
    if isinstance(value, float) == False and isinstance(value, int) == False:
        print('NOT AN INTEGER VALUE FOR XSPEED')
    self.xSpeed = value

#This method changes the y speed of the projectile to whatever is passed into the method
def setYSpeed(self, value):
    if isinstance(value, float) == False and isinstance(value, int) == False:
        print('NOT AN INTEGER VALUE FOR YSPEED')
    self.ySpeed = value
```

```
def displayProjectile(self, screen):
    pygame.draw.rect(screen, colours.red, self.rect)

#This is the class for sight projectiles, which inherits projectile
class SightProjectile(Projectile):
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__(coords, xSpeed, ySpeed, length)
        #This attribute is a list which will contain all of the objects that the projectile has
        #Collided with
        self.collidedObjects = []

#This returns a list of all of the objects which the projectile has collided with
def getCollidedObjects(self):
    return self.collidedObjects

#This is a method that searches through all of the objects in order to check
#If the projectile has collided with it
def searchObjects(self, searchObject):
    found = False
    for object in self.collidedObjects:
        if object == searchObject:
            found = True
    return found

#This method checks if the projectile has collided with any of the objects in the map
#and based on that is sets the visibility as True and if it is a wall then it sets
#The collided attribute as true
def objectCheck(self, allObjects):
    #Goes through all of the objects in the map
    for object in allObjects:
```

```
if isinstance(object, hunter.Hunter):
    rect = object.getHitbox()
else:
    rect = object.getRect()
#Method invoked with that object above passed to check if the projectile has
#collided with it
if self.collideCheck(rect) == True:
    #If it has then it is added to the collided objects list
    self.collidedObjects.append(object)
    #The visibility of that object is set to True so it can be seen
    object.setVisible(True)
    #Checks if the object is a door or a wall and if it is then it has collided so
collided is True
    if isinstance(object, objects.Wall) or isinstance(object, objects.Door):
        self.collided = True

#This method launches the projectile and then runs the other methods in order to check
#what objects the projectile has collided with and returns the list of objects the projectile
#Has collided with
def launchSightProjectile(self, screen, allObjects, coords):
    #Sets the collided objects as an empty list each time the projectile is launched
    self.collidedObjects = []
    #The center of the projectile is set as the coordinates passed
    self.rect.center = coords
    self.launched = True
    #Runs while the projectile has not collided with a wall or door
    while self.collided == False:
        self.moveProjectile(screen)
        self.objectCheck(allObjects)
    #Once it has collided with a wall or door then self.collided will be true
    #So list of all collided objects are returned
```

```
    return self.collidedObjects

class HunterSightProjectile(Projectile):
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__(coords, xSpeed, ySpeed, length)
        self.playerCollision = False

    #This method checks if the projectile has collided with any walls or the player
    #and if it has then it returns the collided attribute as True and
    #if the projectile has collided with the player then the playerCollision attribute is
    #also set as True
    def checkCollisions(self, allWalls):
        for wall in allWalls:
            if isinstance(wall, player.Player):
                rect = wall.getHitbox()
            else:
                rect = wall.getRect()

            if self.collideCheck(rect):
                if isinstance(wall, player.Player):
                    self.collided = True
                    self.playerCollision = True
                else:
                    self.collided = True

    #This method is very similar to the original sight projectile method
    def launchSightProjectile(self, screen, coords, allWalls):
        self.rect.center = coords
        self.launched = True
        self.playerCollision = False
```

```
        while self.collided == False:
            self.moveProjectile(screen)
            #self.displayProjectile(screen)
            self.checkCollisions(allWalls)
        return self.playerCollision

#This metho just returns the playerCollision attribute
def getPlayerCollision(self):
    return self.playerCollision

#This is the sound projectile class which will be used by the hunter to determine the number of walls
between the player and the hunter
class SoundProjectile(Projectile):
    def __init__(self, coords, xSpeed, ySpeed, length):
        super().__init__(coords, xSpeed, ySpeed, length)
        #This attribute will be used to record the number of wall which the projectile has collided
into
        self.wallNum = 0
        self.collidedWalls = []

    #This method checks if the projectile has collided into the player by taking the player's hitbox
as a parameter
    def playerCollision(self, player, screen):
        if self.collideCheck(player.getHitbox()) or self.rect.x < 0 or self.rect.x > 960 or
self.rect.y < 0 or self.rect.y > 640:
            self.setCollided(True)
            return
        self.setCollided(False)
```

```
#This method launches the sound projectile and then returns the number of walls the projectile has
collided into
def launchSoundProjectile(self, screen, coords, map, player):
    self.wallNum = 0
    self.launched = True
    self.rect.center = coords
    walls = map.getWalls()
    while self.collided == False:
        self.moveProjectile(screen)
        self.wallCheck(walls)
        self.playerCollision(player, screen)
    collidedNum = self.wallNum
    self.collidedWalls = []
    #self.wallNum = 0
    return collidedNum

#This method checks if the projectile has collided with a wall in the map and if it has then it
increments the number of walls collided
def wallCheck(self, walls):
    for wall in walls:
        if self.collideCheck(wall) == True and self.alreadyCollided(wall) == False:
            self.wallNum += 1
            self.collidedWalls.append(wall)

def alreadyCollided(self, check):
    for wall in self.collidedWalls:
        if check == wall:
            return True
    return False
```

File: player

```
import pygame
import maps
import objects
import physics
import time
import sprite
pygame.init()

class Player(sprite.Sprite):
    def __init__(self, x, y, img, scale, speed, sprintMultiplier, map):
        super().__init__(x, y, img, scale, speed, sprintMultiplier)
        #Activation area for the player to use to detect levers and hiding places around the player
when the player interacts
        self.activationArea = pygame.Rect(self.x - 32, self.y - 32, self.width * 3, self.height * 3)
        self.hiding = False
        self.sound = None
        #These are the attributes for the levers
        self.maxLevers = None
        self.activatedLevers = 0
        self.sightProjectiles = []
        self.collided = []
        self.sound = None
        self.sprintTimer = 0
        self.interacted = False

    #Displays the player on whatever screen is passed to the method
    def displayPlayer(self, screen):
```

```
if self.getHiding() == False:  
    self.displaySprite(screen)  
  
    #Checks if the W key has been pressed and if it has then it deducts 1 from the player coordinates  
and update  
    #the coordinates of the player  
    def checkForward(self, pressed):  
        if pressed[pygame.K_w]:  
            self.moveForward()  
        else:  
            self.forward = False  
        self.setPlayerCoords()  
  
    #Checks if the S key has been pressed and if it has then it increments the players coordinates by  
1 and updates  
    #the coordinates of the player  
    def checkBackward(self, pressed):  
        if pressed[pygame.K_s]:  
            self.moveBackward()  
        else:  
            self.backward = False  
        self.setPlayerCoords()  
  
    #Checks if the D key has been pressed and if it has then it increments the players coordinates by  
1 and updates  
    #the coordinates of the player  
    def checkRight(self, pressed):  
        if pressed[pygame.K_d]:  
            self.moveRight()  
        else:  
            self.right = False
```

```
    self.setPlayerCoords()

    #Checks if the A key has been pressed and if it has then it deducts 1 from the players coordinates
    and updates
    #the coordintes of the player
    def checkLeft(self, pressed):
        if pressed[pygame.K_a]:
            self.moveLeft()
        else:
            self.left = False
        self.setPlayerCoords()

def leverCheck(self, item, map):
    #Checks if the item is a lever
    if isinstance(item, objects.Lever):
        #Checks if the lever has already been activated
        if item.getActivated() == False:
            self.activateLever()
            #Activates that specific lever so it cannot be activated again
            item.activate()
            print('activated lever')

        else:
            print('LEVER HAS ALREADY BEEN ACTIVATED')

    #This procedure checks if the object is a hiding space
    def hideCheck(self, item):
        #Checks if the item is a hidingspace
        if isinstance(item, objects.HidingSpace):
            self.setHiding(True)
```

```
def interactCheck(self, event, map):
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_e:
            self.interact(map)

#This procedure carries out the interaction function for the player
def interact(self, map):
    if self.hiding == False:
        #Uses a function which returns a list of the walls with items and gives it an identifier
        itemWalls = map.getItemWalls()
        #Goes through each one of these walls
        for itemWall in itemWalls:

            #Checks if the player's activation area has collided with the wall with an item,
            #If the player's activation area has collided with the wall, they are close enough
            if self.activationArea.colliderect(itemWall.getRect()):
                #item is the wall which also contains either a hiding space or lever
                item = itemWall.getItem()
                self.hideCheck(item)
                self.leverCheck(item, map)

    else:
        self.setHiding(False)

#This returns a boolean value based on if the player is pressing the shift button or not
def sprintCheck(self, pressed):
    if pressed[pygame.K_LSHIFT]:
        self.setSprinting(True)
    else:
        self.setSprinting(False)
```

```
#This changes the coordinates of the player whenever it is called, as the map coordinates of the
player are directly linked to the
    #screen coordinates of the player, by changing the screen coordinates (which moves the player
image and rect) it also changes the
        #players location on the other map
def setPlayerCoords(self):
    self.setCoords()
    #Sets the coordinates of the activation area as the center of the hitbox of the player
    self.activationArea.center = self.hitbox.center

    #This method checks if the W, A, S, D, E, or LEFT SHIFT buttons are pressed and then carries out
various actions depending
    #on the button that is pressed
def checkKeys(self, map):
    #Uses a pygame method to check if any key has been pressed and will be true if it is pressed
and false if no buttons
        #on the keyboard are being pressed
    pressed = pygame.key.get_pressed()
    hiding = self.getHiding()
    self.sprintCheck(pressed)
    win = self.checkWin(map)

    #Each of these methods control an aspect of the inputs from the user which in turn controls
the player
    if hiding == False:
        self.checkForward(pressed)
        self.checkBackward(pressed)
        self.checkRight(pressed)
        self.checkLeft(pressed)
    return win
```

```
def checkSound(self, map):
    coords = self.getMapCoords()
    floor = map.getObject(coords)
    if isinstance(floor, objects.Floor):
        sound = floor.getSoundLevel()
        self.setSound(sound)
    else:
        print('NOT FLOOR')

#This sets the sound that the player is making based on the coordinates of the player on the map
def setSound(self, value):
    if value <= 0 or value >= 1:
        print('INCORRECT VALUE')
        return
    self.sound = value

#This returns the sound level for the player
def getSound(self):
    return self.sound

#This returns the value for the number of activated levers
def getActivatedLevers(self):
    return self.activatedLevers

#This function adds 1 to the number of activated levers if not all of the levers have been activated
def activateLever(self):
    self.activatedLevers += 1
```

```
print(self.activatedLevers)

#This returns the max number of levers that are in the game
def getMaxLevers(self):
    return self.maxLevers

#This returns a boolean value of the attribute for if the player is hiding or not
def getHiding(self):
    return self.hiding

#This sets the value of hiding pabased on the parameter that is passed which has to be a boolean
value
def setHiding(self, hiding):
    if hiding == True or hiding == False:
        self.hiding = hiding
    else:
        print('HIDING IS NOT BOOLEAN VALUE')

#This procedure gets a list of all of the levers from the map class and then sets the number of
max levers as the length of that list
def setMaxLevers(self, map):
    levers = map.getWalls('lever')
    self.maxLevers = len(levers)

#This function checks if the player has activated all of the levers and then returns a boolean
value based on that
def checkWin(self, map):
    #function returns the door objects in a list
    doors = map.getDoors()
    #checks if the player has activated all of the levers
    if self.getActivatedLevers() == self.getMaxLevers():
```

```
#As there are multiple doors, this loop goes through each door
for door in doors:
    door.activate()
#Checks if each door has been collided with and carries out action based on that
if self.hitbox.colliderect(door.getRect()):
    #Returns to end game loop
    return True

#This creates the projectiles around the player which will be used for the field of view
def createProjectiles(self):
    #This is the length of each side of the projectile in pixels which will be passed into the
projectiles class
    length = 32

    #As there was no simpler way to create a loop as the projectiles need a specific speed, each
one had to be created individually
    projectile1 = physics.SightProjectile(self.getHitbox().topleft, 0, 5, length)
    projectile2 = physics.SightProjectile(self.getHitbox().topleft, 0, -5, length)
    projectile3 = physics.SightProjectile(self.getHitbox().topleft, 5, 0, length)
    projectile4 = physics.SightProjectile(self.getHitbox().topleft, -5, 0, length)
    projectile5 = physics.SightProjectile(self.getHitbox().topleft, -5, -5, length)
    projectile6 = physics.SightProjectile(self.getHitbox().topleft, -5, 5, length)
    projectile7 = physics.SightProjectile(self.getHitbox().topleft, 5, 5, length)
    projectile8 = physics.SightProjectile(self.getHitbox().topleft, 5, -5, length)

    #As the projectiles were created as individual variables, they must be appended to the
projectiles list individually as well
    self.sightProjectiles.append(projectile1)
    self.sightProjectiles.append(projectile2)
    self.sightProjectiles.append(projectile3)
    self.sightProjectiles.append(projectile4)
```

```
self.sightProjectiles.append(projectile5)
self.sightProjectiles.append(projectile6)
self.sightProjectiles.append(projectile7)
self.sightProjectiles.append(projectile8)

def fov(self, map, screen, hunter):
    #This variable holds a 1D list of all of the objects in the map
    allObjects = map.getAllObjects()
    allObjects.append(hunter)
    #This for loop goes through each of the objects in the list above
    for object in allObjects:
        #This sets the visibility of each object as False so that it appears black on the screen
        object.setVisible(False)

    #This checks if the projectiles have been made yet as they are appended to the list straight
    after they are made
    #and the projectiles only need to be created once as they can be launched many times
    if len(self.sightProjectiles) == 0:
        self.createProjectiles()

    #This for loop goes through each projectile in the self.projectiles list
    for projectile in self.sightProjectiles:
        #This if statement checks whether the projectile has yet to be launched for the first time
        or if it has collided with the wall and finished
        if projectile.getCollided() == True or projectile.getLaunched() == False:
            #If either of the above is true then it sets the collided attribute of the projectile
            as false
            projectile.setCollided(False)
            #and then re-launches the projectile again to check, a list of objects it has collided
            with are returned
```

```
        collided = projectile.launchSightProjectile(screen, allObjects,
self.getHitbox().center)
            #This list is then appended to the self.collided list
            self.collided.append(collided)

        #This nested for loop goes through each object that all the projectiles have collided with
        for projectile in self.collided:
            for object in projectile:
                #This sets the visibility of the object as true so that the player can see it
                object.setVisible(True)
        #The self.collided list has to be reset each time otherwise the previous objects would also be
        set as visible to the player
        self.collided = []

    def getCollided(self):
        return self.collided

    def ready(self, map, screen, hunter):
        self.displayPlayer(screen.getScreen())
        self.fov(map, screen, hunter)
        self.checkCollision(map)
        self.checkSound(map)
        win = self.checkKeys(map)
        if win:
            return True

    def getMovement(self):
        if self.getForward() or self.getBackward() or self.getLeft() or self.getRight():
            return True
        else:
            return False
```

File: screens

```
import pygame
import player
import hunter
import objects
import colours
import time

pygame.init()

#Screen class
class Screen():
    def __init__(self, colour):
        self.colour = colour
        #All lists are 2 dimensional, storing what needs to be displayed
        #and also the coordinates of where they should be displayed
        self.texts = []
        self.images = []
        self.buttons = []
        self.height = 640
        self.width = 960
        #All screens will be the same size
        self.screen = pygame.display.set_mode((self.width, self.height))

    #Procedure to display image
    def displayImg(self):
        imageIndex = 0
        #Runs while both of the counters above haven't gone above the length each list within the list
        while imageIndex != len(self.images[0]):
```

```
#Loads the image
img = pygame.image.load(self.images[0][imageIndex]).convert_alpha()
width = img.get_width()
height = img.get_height()
scaleImg = pygame.transform.scale(img, (int(width * self.images[2][imageIndex]),
int(height * self.images[2][imageIndex])))
self.screen.blit(scaleImg, self.images[1][imageIndex])
imageIndex += 1

#Procedure to render multiple lines of text
def renderMTexels(self, texts, sizes, colours, fonts, textCoords):
    textList = []
    counter = 0
    #Goes though each string of text in the list
    while counter != len(texts):
        #Uses the renderText method to render the text
        render = self.renderText(texts[counter], sizes[counter], colours[counter], fonts[counter])
        #Appends each render of texts to the textList for later
        textList.append(render)
        counter += 1
        #Both the list of rendered texts and the list of coordinates are
        #appended to the texts list within the class
    self.texts.append(textList)
    self.texts.append(textCoords)

#Function to render text
def renderText(self, textName, size, colour, font):
    #Sets font(I/A) and size of the text
    textFont = pygame.font.Font(font, size)
    #Renders the text with anti aliasing(Boolean) and colour (Tuple)
    text = textFont.render(textName, True, colour).convert_alpha()
```

```
return text

#Procedure to display text
def displayText(self):
    textIndex = 0
    #Runs while both of the counters above haven't gone above the length each list within the list
    while textIndex != len(self.texts[0]):
        self.screen.blit(self.texts[0][textIndex], self.texts[1][textIndex])
        textIndex += 1

    #Procedure which accesses the Button class to create a button and then attributes it to this
    #specific class
    def createButton(self, id, image, x, y, scale):
        #Creates an object of type button with the relevant attributes
        button = Button(id, image, x, y, scale)
        #Adds the button to the list of buttons
        self.buttons.append(button)

    #Function which searches and returns a button based on the ID of the button
    def searchButton(self, id):
        #Goes through each of the buttons within the list
        for button in self.buttons:
            #Compares the id of the current button to the
            #one it is searching for
            if button.id == id:
                #returns the button object
                return button

    #Procedure which displays a new screen
    def displayScreen(self):
```

```
#Colours the screen and covers all blitted things
self.screen.fill(self.colour)
self.displayImg()
self.displayText()

#Procedure which appends all image based things into the images list
def addImages(self, images, Coords, scale):
    self.images.append(images)
    self.images.append(Coords)
    self.images.append(scale)

#Procedure which sets colour of the screen
def setColour(self, colour):
    self.screen.fill(colour)

def getHeight(self):
    return self.height

def getWidth(self):
    return self.width

def getScreen(self):
    return self.screen

#Button class
class Button():
    def __init__(self, id, img, x, y, scale):
        #Identification needed for each button so that they can
        #be differentiated and easier to find
        self.id = id
        #Loads the image
```

```
self.img = pygame.image.load(img).convert_alpha()
#Gets the width and height of the img in pixels
width = self.img.get_width()
height = self.img.get_height()
#Transforms the image using a scale
self.transformedImg = pygame.transform.scale(self.img, (int(width * scale), int(height * scale)))
#Gets the rectangular area of the image
self.rect = self.transformedImg.get_rect()
#Is the top left of the image
self.rect.topleft = (x, y)
self.clicked = False

def draw(self, screen):
    #draws image on the screen
    screen.blit(self.transformedImg, (self.rect.x, self.rect.y))

def clickCheck(self, screen):
    self.draw(screen)
    hasClicked = False
    pos = pygame.mouse.get_pos()
    #Is mouse cursor colliding with the rectangle of image
    if self.rect.collidepoint(pos):
        #Checks if the mouse has been pressed and has already been pressed before
        if pygame.mouse.get_pressed()[0] == 1 and self.clicked == False:
            #Set to true so that button is only registered once with one mouse click
            self.clicked = True
            hasClicked = True
    #If the user is not pressing the mouse button it is reset so that the user
    #can use it again
    if pygame.mouse.get_pressed()[0] == 0:
```

```
        self.clicked = False
    #Returns whether the mouse has been clicked or not
    return hasClicked

class GameScreen(Screen):
    def __init__(self, colour):
        super().__init__(colour)
        self.colour = colour
        self.timer = 0

    def displayRect(self, rect, colour):
        #Draws a rect on the screen
        pygame.draw.rect(self.screen, colour, rect)

    def displayGameScreen(self, map):
        #Displays the screen and the map
        self.timer = time.time()
        self.displayScreen()

        for row in map:
            for object in row:
                if object != 0:
                    self.displayRect(object.getRect(), self.checkObjectColour(object))

    def checkObjectColour(self, object):
        if object.getVisible() == True:
            if isinstance(object, objects.Wall):
                item = object.getItem()
                if item == None:
                    return colours.darkGrey
            elif isinstance(item, objects.HidingSpace):
```

```
        return colours.blue
    elif isinstance(item, objects.Lever):
        if item.getActivated() == True:
            return colours.green
        else:
            return colours.red
    elif isinstance(object, objects.Floor):
        type = object.getType()
        if type == 'carpet':
            return colours.navy
        elif type == 'concrete':
            return colours.lightGrey
        elif type == 'wood':
            return colours.lightBrown
    elif isinstance(object, objects.Door):
        if object.checkDoorActivation() == True:
            return colours.white
        else:
            return colours.darkBrown
    else:
        return colours.black
```