* **Text in red is added based on feedback**
* **Text in Green is added based on feedback due 24th march (yet to be finished)**

**Problem Identification**

Year 12 and 13 students have been working extremely hard for the A-levels. Where year 13 students have been studying rigorously for exams in half a year, Year 12 students have been spending hours at home studying current topics to not be left behind and clueless by the endless amounts of work to be set for them in the future. However, rest is essential for optimal performance in class and for memory retention. As a result of this, I have come up with a solution that will keep students in peak- condition for class and give them beneficial relaxation to be able to keep up with the struggles of A-levels. 'ASMR' is a tingling sensation in response to a stimulus such as audio. 'ASMR ' has become a very popular method for relaxation. Its peak was during lockdown when people were stressed due to lack of essential products and money. It was used with no regard to demographics and was available to all. Without a doubt, creating an ASMR game will be extremely successful in allowing students to relax and have a flawless experience in sixth form/college.

**Why is it suited to a computational solution**

The solution is suited to a computational solution because it will require software that can only be run by a computer machine. It will use software to produce graphics that are calming to the eye and are also found pleasing to the players. Furthermore it will require audio to immerse the players and perfect the 'ASMR experience'. This is necessary because 'ASMR' requires a stimulus such as audio or visual graphics as it needs to provide a satisfying feeling to the player otherwise it will lose its effect at relaxing the audio and only tire out the player. A keyboard would be required in order to complete the aim of the game. Headphones are optional, but would improve the quality of the game and user experience. There won’t be any need for other input devices such as a mouse as the user will control the game with a keyboard. To summarise, the environment in which the game is to be played is on a computer. This means it will need software. This makes a computational solution the only option. Without a computational solution, I will be unable to complete the tasks to create the game as no other solutions are compatible with this problem.

**Problem Recognition(rewritten)**

Making a seamless, interesting, and well-rounded game that focuses on creating a stimulus is the key challenge. Making something that users can interact with is the fundamental issue. Then, I can create the game by solving each issue separately until it offers the user a satisfying, all-around fun experience. I need to use problem recognition throughout this to avoid overlooking a vital element of the game that might be essential to its progress. This may involve forgetting to add key elements to the game, or making it harder to debug if i find errors in my code.

**Problem Decomposition:**

**Decomposition is the process of dividing a huge difficulty into a number of smaller, simpler difficulties. This will benefit me because it will cut down on errors and make it simpler for me to comprehend the issue at hand. By segmenting it into multiple parts, I can incorporate this into my game. Characters, objects, movement controls, images, and music are a few examples. These are all simpler to solve than finishing the entire game at once. Decomposition will be essential to the development of my game because it helps me allocate my time effectively. My criteria to achieve this would be to:**

Create an environment that allows for a satisfying event to occur (drawing a perfect circle, filling a glass of water to the top without spilling any, etc).

Add audio to stimulate 'ASMR' and create an immersive experience.

Create different levels/scenarios that replicate the event/action to be repeated.

**Divide and Conquer:**

Creating the game in one go is extremely difficult and would not be an over-exaggeration to call it impossible for someone who is still learning how to code. However, creating sub tasks to create a modular program makes coding the came very doable, even for a beginner/intermediate coder as myself.

**Abstraction:**

**Abstraction is the process of removing superfluous information from a problem in order to concentrate effort on key issues. Additionally, it stops superfluous code from interfering with the game's more crucial features. I'm going to employ abstraction in my project. I'll make use of it by getting rid of the less important elements of my game. For instance, my game's characters and items don't have to be realistic. Making them realistic would require too much time and complicate game play, neither of which are advantageous. Making the game 2D is another. With that being said;** Creating detailed objects or a background is not necessary. The focus should be the stimulus and the audio. The audio should be clean with no background noise and the video graphics should not be laggy, but smooth. The game will be made suited for a computer and not a mobile device, although a mobile version can be replicated as touch would work just fine with the concept of the game. The input keys would not require a mouse, but a keyboard. There won't be need for many keys as there will only be simple actions that make the game easier to play and stress-free.

Thinking Ahead:

Planning ahead is essential for more effective time management. One approach I'll plan ahead for my project is to establish prerequisites. Pre-conditions can be specified to make programme components more reusable, eliminate pointless checks, and make programmes simpler to debug and maintain. Pre-conditions can be specified to make programme components more reusable, eliminate pointless checks, and make programmes simpler to debug and maintain. I'll also plan ahead by employing programming standards when I code. When analysing code later, programming standards make it simpler to debug and maintain. I'll apply programming standards, for instance, by documenting inputs and outputs, utilising modularity, and adding comments as I code.

Thinking Concurrently:

Concurrent processing refers to the simultaneous execution of instructions by several processors. Concurrent processing will be used in my game to ensure smooth operation because it will provide users the finest experience possible and keep them interested for longer. It also allows for more simultaneous events, making the game more complex and hence more interesting. Keeping score while the user is playing or creating new adversaries while the user is battling existing ones, for instance

**Interview:**

**Interview questions:**

I will outline some of the key questions that I will ask each stakeholder, but during the interview I may ask follow up questions or ask them to elaborate on certain answers. The questions will find their opinion on the software and how they would like to use it.

I have picked my stakeholders based on my demographics.My demographic includes students in a level of education higher than primary school, those who enjoy playing games, any level of gamer (casual or hardcore) and those who are going through stress. However, my game does not only apply to students, and can be used by those in primary school, and it can be played just for fun instead of relieving stress.

I have created a questionnaire on google forms in order to ask some questions for my stakeholders in order to find their opinion on stress and how they deal with it. I have also asked them questions based on my niche (ASMR) in order to find out more information on how I can use it in my game.

My questions for my stakeholders were:

1. Which of the following applies to you? (to find out which type of education they are involved in)
2. How are you finding it?
3. Would you say that you suffer from stress?
4. Do you enjoy playing games?
5. ​​Have you heard of ASMR?
6. ​​If you answered yes, do you think it is useful for reducing stress?
7. Have you played an ASMR game before, if so, what was the game like?
8. What was a feature you disliked about the game?
9. What was a feature you liked about the game?
10. Do you have any suggestions for how my game should be like?
11. What category should the game be? (e.g. world-builder, strategy, puzzle, etc.)

I have asked these questions specifically in order to find out more about the needs of my stakeholders and the position they are currently in so I can create a solution for all.

Interview answers:

1. All my stakeholders were either in college or sixth form.
2. My stakeholders responded either moderate or difficult.
3. A majority say they feel stressed sometimes due to college/sixth form, however, only a small percentage said that they genuinely suffer from it.
4. All my stakeholders were gamers, however, my demographic are gamers so this is not an issue.
5. All of my stakeholders had heard of ASMR.
6. All of my stakeholders had said it was useful in reducing stress and that they have used it themselves in order to relieve their own stress before.
7. All of my stakeholders had played an ASMR game before on their mobile devices. The games had many satisfying sounds, a calming background, and easy game mechanics which meant that they were stress-free.
8. Some disliked the background music,whilst others found the game boring after a short period of time.
9. All enjoyed the simplicity of the game and found it fun and relaxing, even though it only lasted a short period of time for some
10. I have been suggested to make an enjoyable game based off of extremely popular and addicting ASMR games such as “Helix Jump” and “Jump shot” which have millions of downloads on the app store and google play store.
11. There were a variety of answers, however, the most popular was puzzle/strategy. This is a much simpler game to design and is easier to randomise than a worldbuilding game. I can take inspiration from a variety of games that require strategy and/or puzzles such as jelly fill or cat escape. Games such as jelly fill are similar to tetris which involves strategy and is a puzzle game.

**Analysis** :

My stakeholders are in the age group of 16-17 year olds, and my demographic also includes gamers (any type, e.g. casual, intense, etc). All of my stakeholders have already used ASMR as a way of relieving stress through videogames and have told me that it was very successful. However, an issue that was found was that some games were monotonous and lost their purpose fairly quickly. This would mean that my goal is to create a game that is addictive so the player never feels bored and the game meets its aim which is to relieve stress.

In order to create a game that appeals to my stakeholders, the requirements that must be met are:

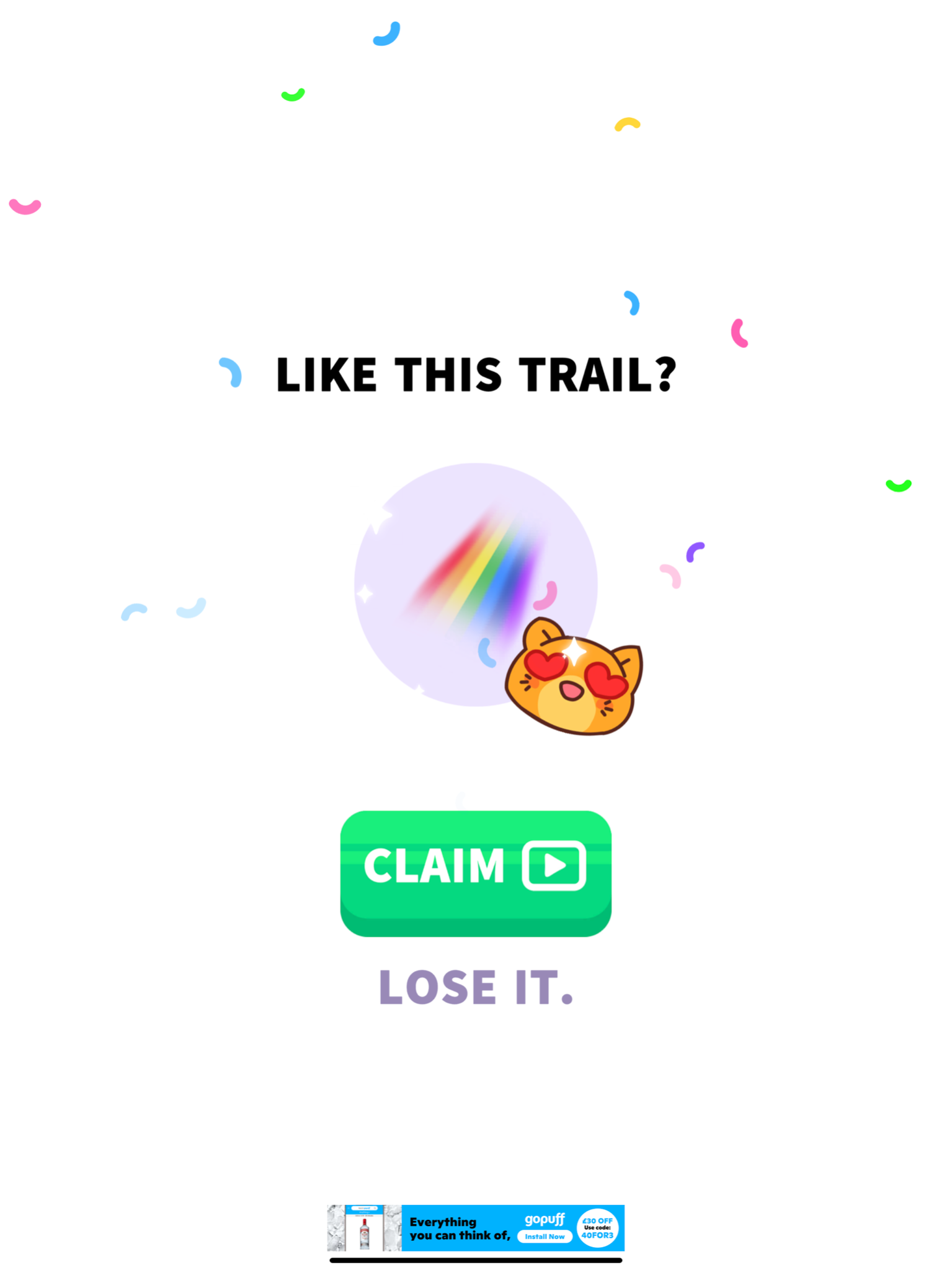
> A simple game that doesn’t have complex mechanics and an easy to achieve goal;

> Non-repetitive gamerounds - this is to ensure that the player doesn’t get bored playing the same thing. This could include a levelling system which increases difficulty:

> Different background music - so the player doesn’t get irritated hearing the same song repeatedly. Could sync a radio to the game or download some copyright-free songs;

> The genre of the game will be puzzle/ strategy. This genre was common amongst my stakeholders because it is simple and gives quick satisfaction. This makes the game addictive and hence why it is popular.

**Research:**

**Cat Escape**



**Overview:**



The game’s objective is to get past the

Guards as cat and leave through the

Green door. This is a mobile game that

can easily be replicated on pc using a

Keyboard as input. The requirements

Are relatively light as the graphics are

simple and any modern day computer

could run the game with ease

**What can i use from this:**

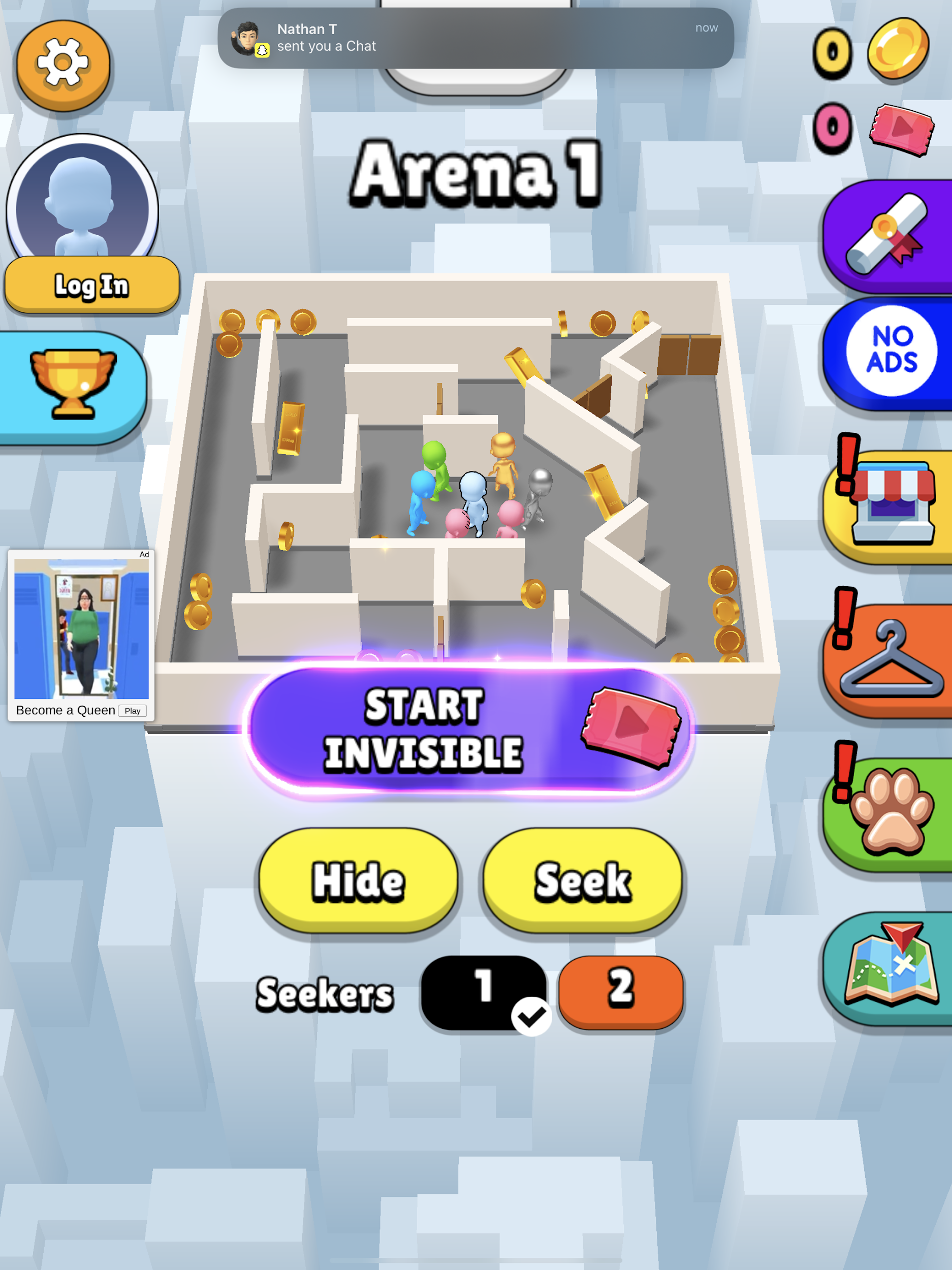
The game has a levelling system where

The objective becomes harder and

Harder to complete. This ensures the

Game doesn’t become monotonous and

keeps the user engaged. The game also has a power system which allows the user to use a specific power in a round. The chance of the power spawning is random and makes the level easier for the user if they so choose. This keeps the game interesting and also unique, hence making the game fun for the user. Similarly to Helix Jump, this game offers cosmetics which allow the user to customise the cat. Some cosmetics correlate to the level which you are on, this allows the users to show off their achievements through cosmetics.



**Overview:**

Hide n. Seek is a similar game to Cat escape in the

sense that you are escaping from someone.

However, what’s different is that it's one/two seekers,

and several other people playing. The way to win

Is to not be captured until the timer runs out. There is an

Option for two seekers or one which gives the user

Choices which are always appealing to the user. Quite

obviously, two seekers is harder than one seeker, and

That is the only way to make the game harder. You are

Also given the option to seek in case the player gets

Bored of hiding. This allows that game to be more fun to

The user as they now have many choices. A fun addition

The game has is side quests. This may

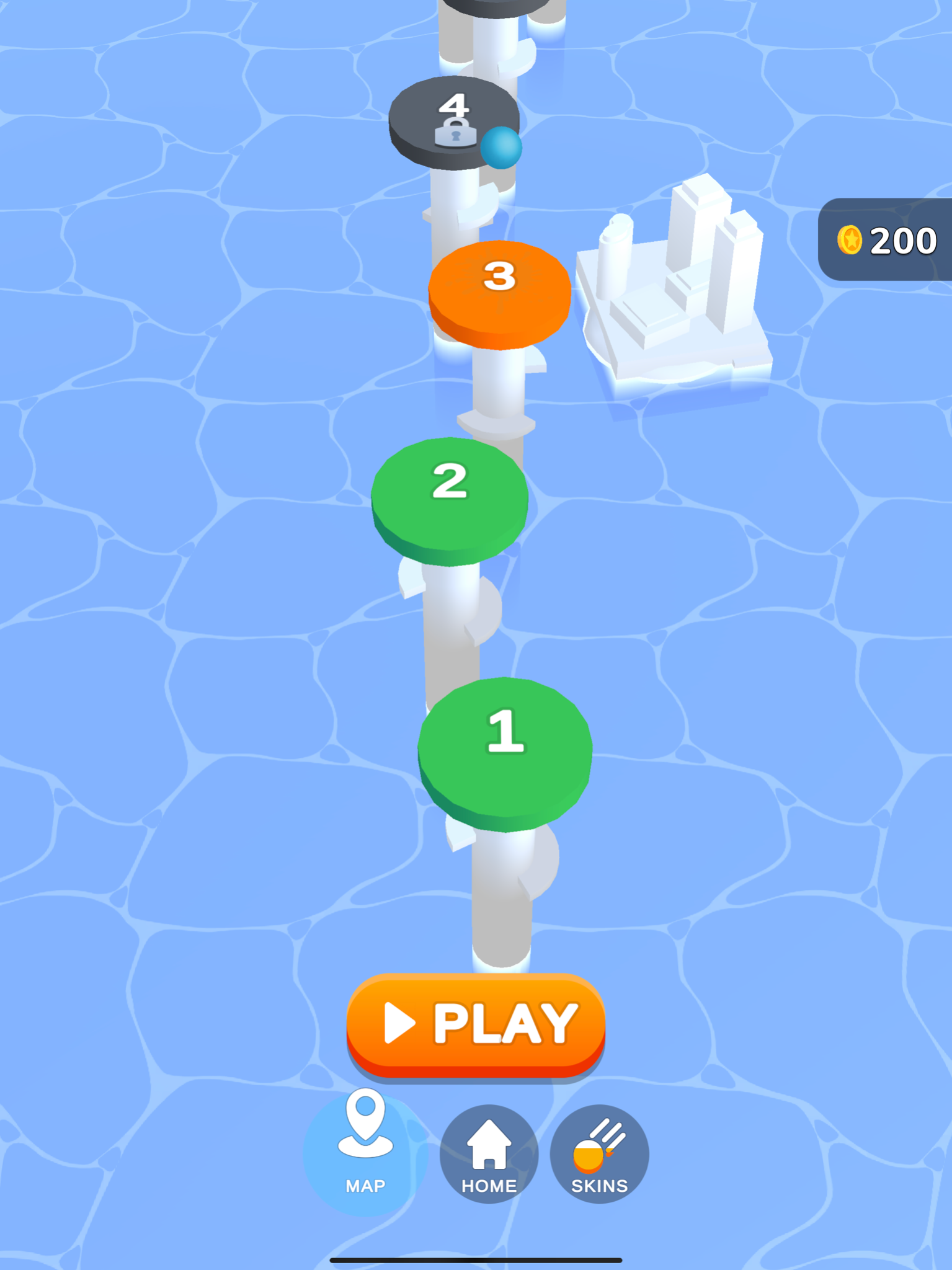
Include outlasting a certain amount of players or

Playing for a certain period of time. These side quests

Offer rewards such as cosmetics and pets. These side quests are updated and are constantly giving the user something new to do which makes the game fun and never monotonous to the player.

**What can I use from this:**

The user of the timer as a seeker gives the user a sense of urgency which stimulates adrenaline. This would help make my game more fun for the user as they know they only have a certain amount of time to complete a level and make the game more challenging. In addition to this feature, I also liked the side quests feature. Although I won’t be updating the game frequently, I can set a goal such as, “collect 200 coins to unlock a new skin”. This will give users a quest to work on so they won’t feel bored playing another escape game. Giving users a challenge and a reward will stimulate their dopamine so they will feel accomplished and distress them from any problems they may have been facing, hence why i feel this feature would be important for my game.



**Helix Jump:**

**Overview:**

The objective of the game is to travel through the

Gaps and avoid the hazardous bits. There are

Different levels and a scoreboard that tracks the

Amount of points you receive until you fail. This

Carries on even if you have finished the level.

Also, a feature in this game similarly to Cat -

Escape is cosmetics. This gives the user motivation

To play the game in order to receive cool

Cosmetics which they can use in game.

**What I can take from this:**

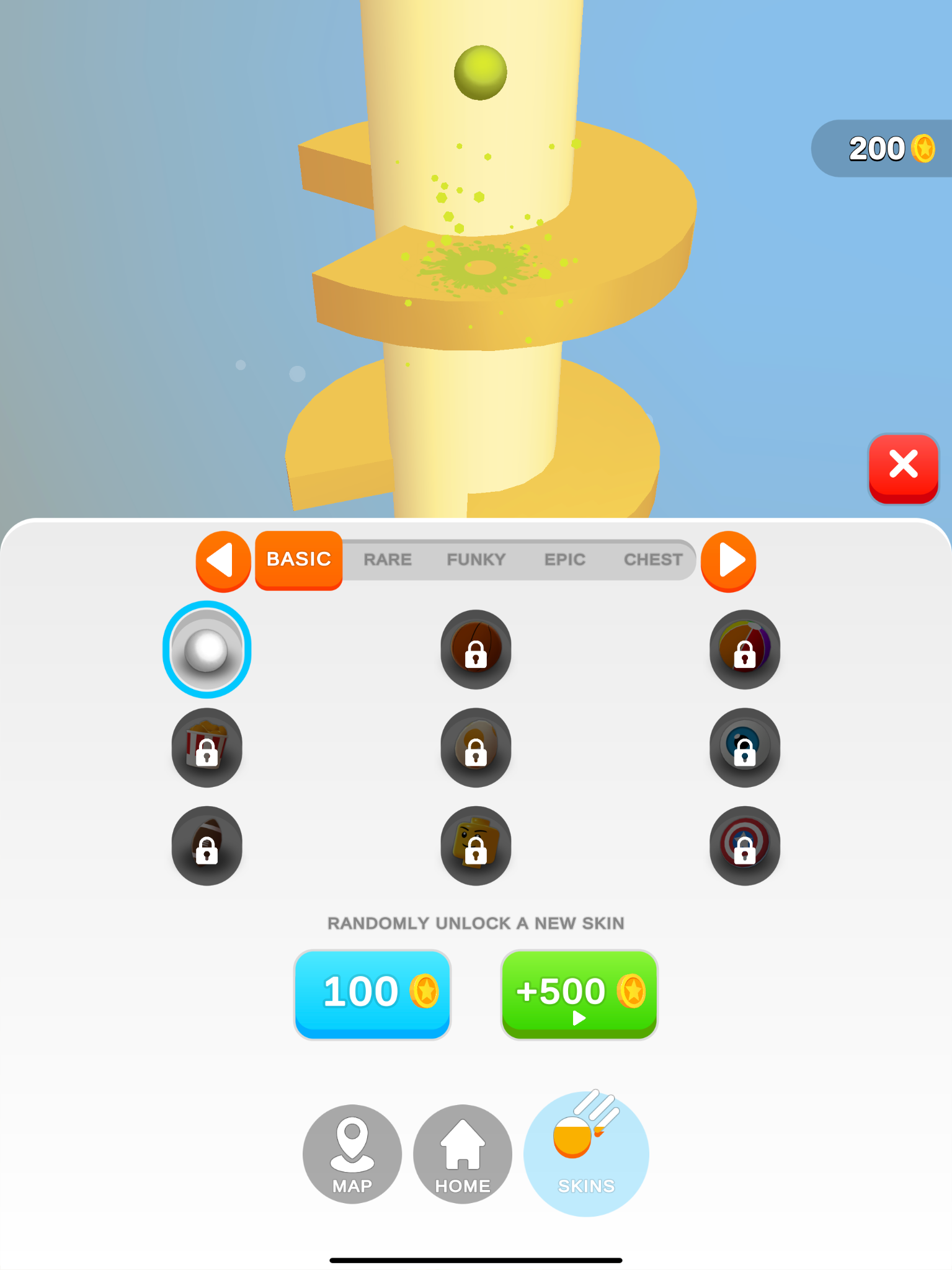
I can use the cosmetics in order to make the users

Want to play longer so they can receive the

Rewards. Also, i could use the scoreboard system

To get competitive players hooked onto the game

In order to attempt to beat their highscore. This would increase my demographic and allow my game to be more open to the different types of gamers.



**Fundamentals of my game:**

My game will be similar to Cat Escape as I found it really inspirational during my analysis phase. I liked the concept of powers and cosmetics and the strategic element of the game. However, it wasn’t very stressful and I had a good experience relaxing whilst playing Cat Escape. This is fundamental because without a strategic element, the game would become boring very quickly and won’t serve its purpose as a stress relieving game. To do this, I will need to simulate a basic maze with an entrance and an exit for the scenario. My character will need to be able to walk up, down, left and right. I will need obstacles such as guards to present a challenge to the game. I will also need to use contrasting, bright colours in order to clearly distinguish different objects from each other. This is important as a player may judge the game based on the style of the game (the way it looks), and if the design quality is poor, then a lot of players may not even pick up the game to try. Hence, I need the game to look top quality on top of enjoyable.

**Limitations of my game:**

My game will be created in pygame. This means that unlike Cat Escape, my game will not have 3D graphics. This will heavily affect the visual satisfaction of the game as a 2D game is very limited to how it can look. 2D games tend to refer to retro games, which in some cases may bring back a feeling of nostalgia which leaves the player satisfied, but due to my demographics being young students this is very unlikely. Another limitation of my game is that the user inputs will be limited to keyboards. There is no need for a mouse and furthermore it will not be adapted to mobile or controller. This is limiting as some players may feel more comfortable with a controller or playing on their mobile devices, ultimately making the game less enjoyable. A third limitation to my game is that it is coded in pygame. Pygame limits the amount of details that can be present in a game, thus restricting me to make a satisfactory game similar to Cat Escape and making it more challenging for me to meet my fundamentals.

**Hardware and Software requirements:**

Hardware:

A PC/laptop - the game is a computer based game, hence a pc/laptop is required. Specs of the computer isn’t a worry as it will be a simple coded game. Furthermore, most modern computers and laptops are capable of running high fps games. For input devices, a keyboard is required as it is the only input device required to play the game, using only a few keys in order to control the character, etc. For output devices, a monitor is required in order to display the game so the player knows what to do.

Since the game is 2D, A computer running windows 7+(or mac or Linux equivalent os), 2GB of ram and no graphics card is required to run the game. This is the minimum specs, however, to run a 2D game, it is recommended to use better specs than this to run the game with a high frame rate and vibrant colours.

Software:

An operating system that can run pygame - an operating system such as macOS,windows or linux is required to run the game. The operating systems I listed are able to run python.

Python interpreter - This may include an online python interpreter such as replit or downloaded software such as python idle.

**Success Criteria:**

| Criteria | How to evidence |
| --- | --- |
| A title screen | Screenshot the title screen |
| Simple design (not too many buttons or complicated settings), with bright and contrasting colours | Screenshot the window, showing the design and explaining the thought process behind the design. |
| Text on top of the buttons so that users can navigate with ease | Screenshot the buttons on the window |
| A settings menu to control the sfx, etc.  It will leave the home screen and must have a back button. | Screenshot the button and its page. |
| A tutorial/information page which explains the mechanics of the game | Screenshot the information explaining the game. |
| Character moves  Guard moves | Evidence testing that the main feature works may be shown in a short video. |
| Obstacles are actual obstacles and can’t be ignored | Evidence may be shown in a short video |
| Cosmetics page that actually shows different styles of the character in game | Screenshot and may be shown in a short video |
| Levelling system | Screenshot the page that shows the stage the user is on and the difficulty of the stage in the game. |

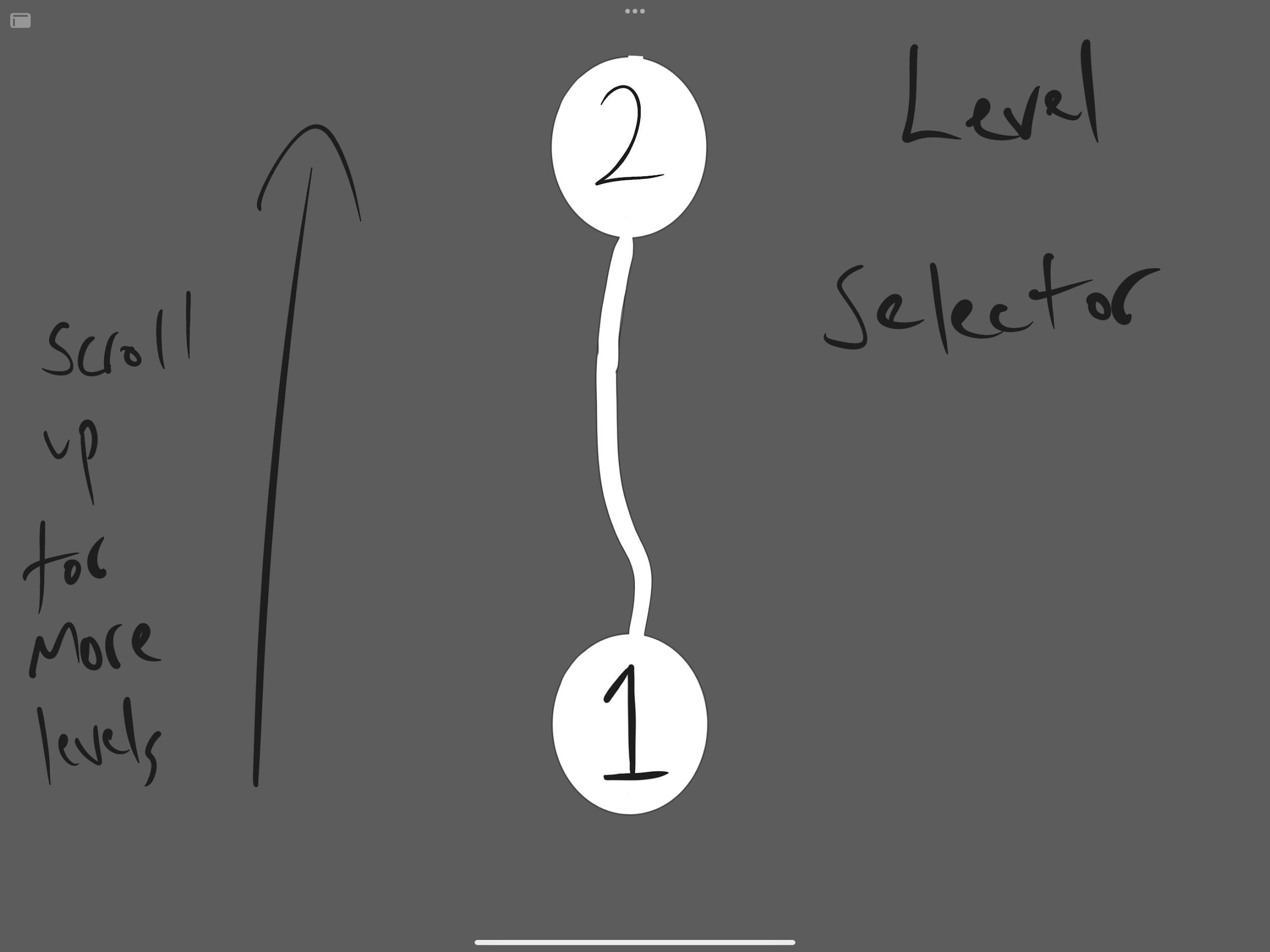
**Design:**

**Overview:**

**My game is inspired by escape games such as escapists and cat escape. The main objective is to exit the building without getting caught. There will be many obstacles such as guards, but there will be many ways to escape thanks to the addition of powers. The game will have a starting screen. From there, the levels system can be accessed to play the current level the player is on or go back another level. The other two options are the skins page and the menus page. The skins page will allow the user to customise their character and see their progress towards unlocking new skins. The menus page will allow the user to control their sound effects volume without it affecting their computer audio.**

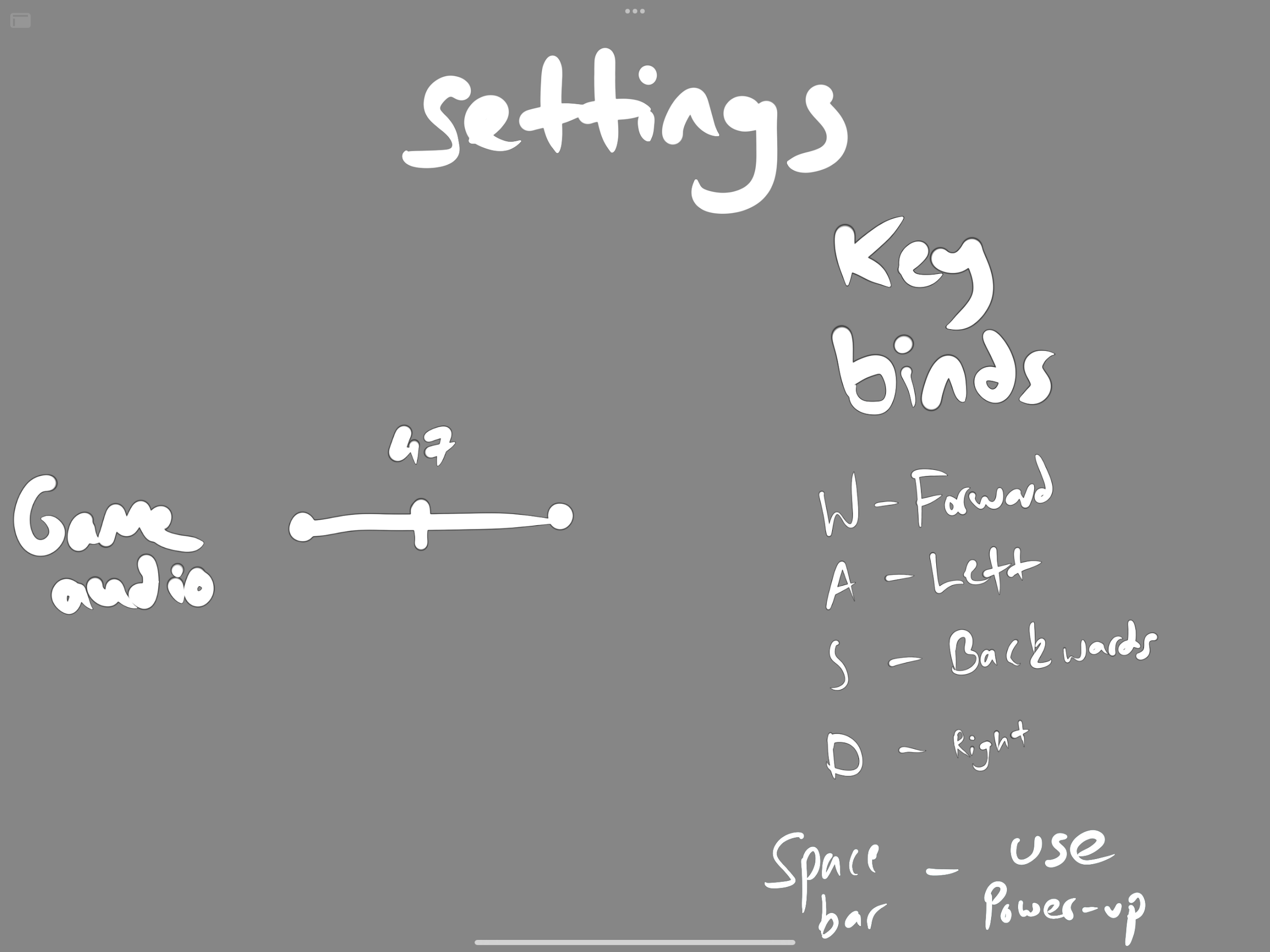
****

**This is the rough design of the title screen. This allows the user three options, Play, skins and settings. It will still be possible to close out the application using keyboard shortcuts or pressing the close button on the top right of the application. Pressing the play button will bring the user to the levels page which is an interactive way for the user to play harder levels after unlocking them. The levels are also re playable in case the user wants to go back to a certain level.**

****

**This is a rough design of the levels page -^**

**The settings page will allow the user to modify their game audio, so they can listen to background music from a third-party app without having to turn off the device's audio and not be able to listen to anything. The keybinds are also shown so the user will know what keybinds to use. However there will be a tutorial so its not the main use of the keybinds page. The main use of it is that it allows the user to modify the keybinds to however they please. This allows more user to play if their keyboard has certain buttons damaged or if it’s more comfortable to use different keybinds.**

****

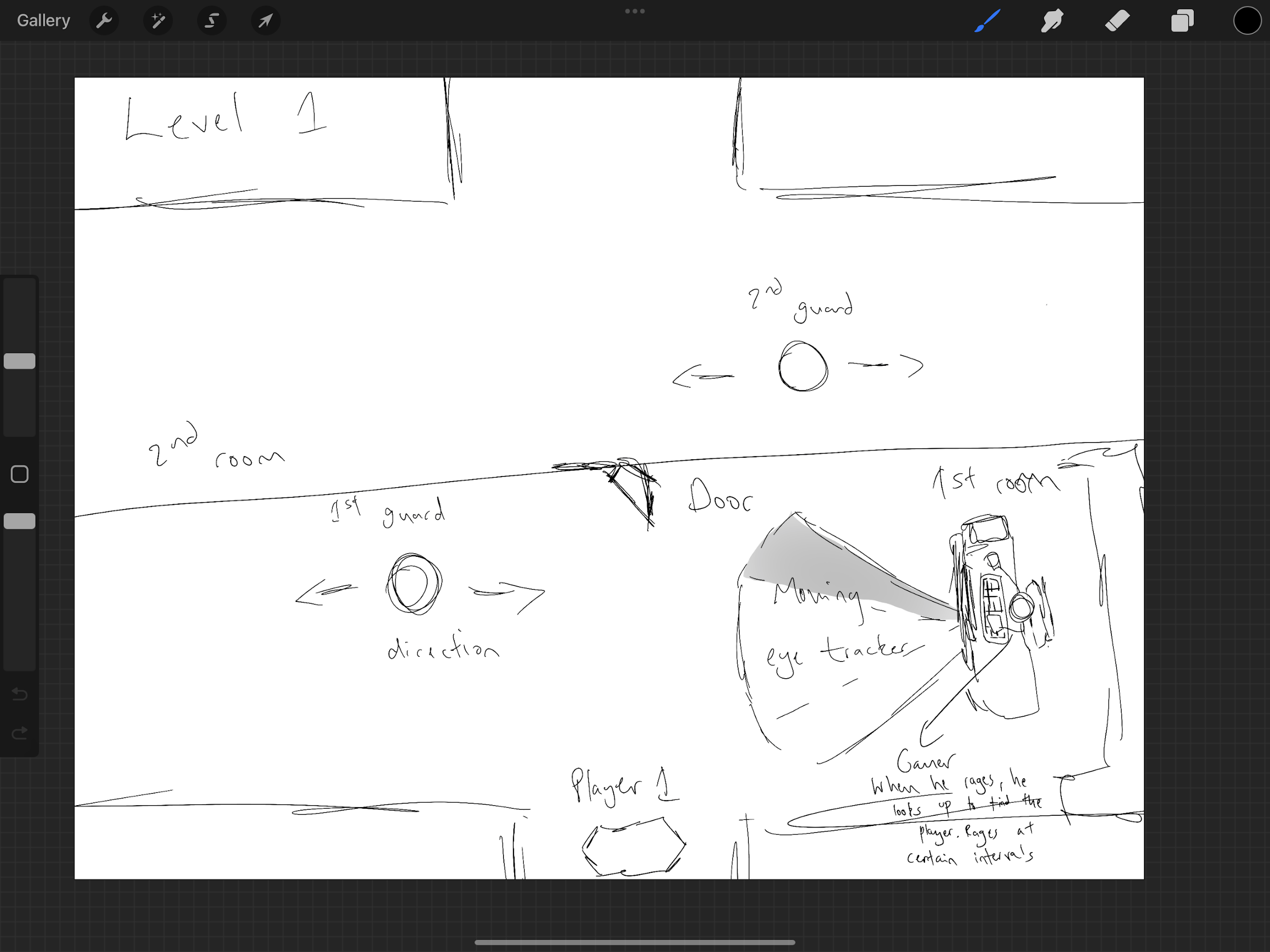
**This is the rough design of the settings page -^**

**The skins page allows the user to customise their character whilst they play. These may be unlocked every few levels or bought with the coins they earn as they play the game.**

****

**This is the rough design for the skins page -^**

**As seen from the image above, the skins will be 2d faces. These are two concepts i came up with, however, i plan on creating many more as they are not time consuming and are relatively easy to complete. They also add a more fun aspect to the game which is why i plan on creating many. It may be a goal for the user to collect all the skins or buy the ones they like. I plan on creating a special skin only for those who collect all skins. This would be a constant goal for some which would create interest for the game.**

**I have 1 level designed already. This level will be the tutorial level and its only one floor and quite easy to complete. However, I haven’t made it too easy as some players may find it boring off the bat and not play.**

**This is the rough design for level 1(tutorial) -^**

**As seen in the image above, there are two types of npc’s, Guards and a gamer. The gamer looks up as he gets upset with his game at random intervals between 3 and 6 seconds. The guards move horizontally across the screen and have a similar eye tracker mechanic as the gamer. The tutorial will explain the mechanics and the objective of the game to players, and as mentioned in the levelling page, the level is re-playable in case a player wants to go over the tutorial again. In the tutorial, there aren't any powers as the levels are too simple for powers, I plan on adding them later on in the game when the levels become more challenging.**

**Powers**

**Powers grant the user special abilities, a few I had in mind were invisibility or super speed. These powers would aid the user in completing a level. Whilst using these powers, I will make levels that won't be possible to complete without the powers, so it won't make the powers boring as the levels would be too easy to complete with them. They will have a certain duration before they are unusable. For example, invisibility will have a timer of 4 seconds. This will make them more stable and the game still enjoyable.**

**Algorithms:**

**Player Movement Algorithm**

**The algorithm for the user’s character’s movement allows the character to move up, down, left and right. The character will have to be able to move in these four directions in order to avoid obstacles and avoid detection from the guards. There is a speed variable which for now is set to one. However, one of the power ups I will add will be super speed. This will allow users to travel a further distance. For example, if speed is set to 4, the player will travel 4 units instead of 1 unit.**

**The algorithm uses pygame’s KEYDOWN library which uses keyboard inputs to determine outputs**

**function player\_input():**

**for event in pygame.event.get():**

**if event.type == pygame.KEYDOWN:**

**if event.key == pygame.K\_LEFT:**

**# press left key**

**player.move\_left()**

**elif event.key == pygame.K\_RIGHT:**

**# press right key**

**player.move\_right()**

**elif event.key == pygame.K\_UP:**

**# press up arrow key**

**player.move\_up()**

**elif event.key == pygame.K\_DOWN:**

**# press down arrow key**

**player.move\_down()**

**class Player:**

**def \_\_init\_\_(self, x, y, width, height):**

**self.x = x**

**self.y = y**

**self.width = width**

**self.height = height**

**self.speed = 1**

**function move\_left(self):**

**self.x -= self.speed**

**function move\_right(self):**

**self.x += self.speed**

**function move\_up(self):**

**self.y -= self.speed**

**function move\_down(self):**

**Self.y +=self.speef**

**Guard Movement Algorithm**

**Since the guards can only move on a horizontal plane, their only movement will be left and right. Guards will move until they reach the other side of the map/ a wall/ obstacle and then turn around to walk towards the other side of the map/to the nearest obstacle. Pygame.rect is for storing rectangular objects with dimensions and its positioning. This can be used to hold information of the surroundings and obstacles of the rooms. For example, walls, bookshelves, etc.**

**class Guard:**

**def \_\_init\_\_(self, start\_pos, end\_pos, speed):**

**self.start\_pos = start\_pos**

**self.end\_pos = end\_pos**

**self.speed = speed**

**self.direction = 1**

**# 1 = right, -1 = left**

**self.current\_pos = self.start\_pos**

**Function update\_position(self):**

**if self.direction == 1:**

**self.current\_pos += self.speed**

**if self.current\_pos > self.end\_pos:**

**self.current\_pos = self.end\_pos**

**self.direction = -1**

**else:**

**self.current\_pos -= self.speed**

**if self.current\_pos < self.start\_pos:**

**self.current\_pos = self.start\_pos**

**self.direction = 1**

**def get\_rect(self):**

**return pygame.Rect(self.current\_pos, self.y, self.width, self.height)**

**Guard Detection Algorithm**

**The algorithm for detection for the guards will be a rectangle area that has a length of 5 units and a width of 3 units. This gives the guards a feature similar to eyesight. This function is used to catch the user if they enter the range of the rectangle.**

**Function check\_guard\_detection(player, guards):**

**for guard in guards:**

**guard\_rect = pygame.Rect(guard.current\_pos, guard.y, guard.width, guard.height)**

**detection\_range = pygame.Rect(guard.current\_pos, guard.y - 5, 5, 10)**

**if detection\_range.colliderect(player.get\_rect()) and guard\_rect.colliderect(player.get\_rect()):**

**if player.current\_pos > guard.current\_pos and guard.direction == 1:**

**return True**

**elif player.current\_pos < guard.current\_pos and guard.direction == -1:**

**return True**

**return False**

**Obstacle Detection Algorithm**

**The check\_obstacle\_collision function examines whether an obstacle is present at the current player location (player\_x and player\_y) on the level map. It returns True if there is an obstruction, meaning that the player cannot travel in that direction.**

**function check\_obstacle\_collision(player\_x, player\_y, obstacles):**

**for obstacle in obstacles:**

**obstacle\_x, obstacle\_y, obstacle\_width, obstacle\_height = obstacle**

**if (player\_x + PLAYER\_WIDTH > obstacle\_x) AND (player\_x < obstacle\_x + obstacle\_width) AND (player\_y + PLAYER\_HEIGHT > obstacle\_y) AND (player\_y < obstacle\_y + obstacle\_height):**

**return True**

**return False**

**Power-Up Algorithm**

**The algorithm is used to alter the player’s base stats such as speed and invisibility as shown below.As mentioned previously, the user’s movements algorithm contained a speed variable which allowed the user to move 1 unit. What the power up algorithm below has done is increase the player’s speed by 1 so that the player now moves 2 units instead of 1 unit**

**Function activate\_power\_up(power\_up, player):**

**if power\_up.type == "speed":**

**player.speed\_x += 1**

**player.speed\_y += 1**

**elif power\_up.type == "invisibility":**

**player.invisibility = True**

**Power-Up Detection Algorithm**

**The Power-Up Detection Algorithm has been set up to detect when the user has collided with the power-up laying on the ground. If the player collides with it, then it is ‘activated’ and the player will receive a power-up.**

**Function check\_power\_up\_collision(player, power\_ups):**

**for power\_up in power\_ups:**

**if player.collides\_with(power\_up):**

**activate\_power\_up()**

**Level Loading Algorithm**

**When a player chooses to play or restart a level, the level loading algorithm is in charge of loading the game level. It involves initialising all required game objects and variables as well as reading the level data from a file.**

**The load\_level function takes in a level\_number parameter which indicates the level to load. The function constructs the filename for the level data file based on the level number and opens the file for reading. It then reads the level data from the file and initialises the necessary game objects and variables using the data.**

**Function load\_level(level\_number):**

**level\_filename = "level" + str(level\_number) + ".txt"**

**with open(level\_filename, "r") as file:**

**level\_data = file.read()**

**player\_position = level\_data.player\_position**

**obstacle\_positions = level\_data.obstacle\_positions**

**guard\_positions = level\_data.guard\_positions**

**level\_map = level\_data.level\_map**

**return player\_position, obstacle\_positions, guard\_positions, level\_map**

**Level Transition Algorithm**

**When a player finishes a level, the player needs to transition from one level to the next. The function takes in the level number and increments by 1 to give the next\_level\_number. If the user has completed all the levels, they are returned to the home screen where they can repeat previous levels, access settings or any other subroutines they need to access.**

**function transition\_to\_next\_level(current\_level\_number):**

**next\_level\_number = current\_level\_number + 1**

**if next\_level\_number > MAX\_LEVELS:**

**return\_to\_home\_screen() # returns to home screen if no more levels are left**

**else:**

**level\_map, player\_x, player\_y, obstacles, guards, powers, exit\_position = load\_level(next\_level\_number)**

**DISPLAY\_LEVEL\_TRANSITION\_SCREEN(next\_level\_number) # show transition screen**

**RETURN level\_map, player\_x, player\_y, obstacles, guards, powers, exit\_position**

**Player’s Base Stats Algorithm**

**The player's base stats are required for many essential features in the game such as power-ups and movement. The player’s base stats hold important properties such as speed and the player's dimensions.**

**Class Player:**

**Function \_\_init\_\_(self):**

**self.speed = 1**

**self.width = 2**

**self.height = 2**

**self.invisibility = False**

**self.skin = "none"**

**self.score = 0**

**Function set\_speed(self, new\_speed):**

**self.speed = speed**

**Function set\_skin(self, skin):**

**self.skin = skin**

**Function set\_invisibility(self, invisibility):**

**self.invisibility = invisibility**

**Function set\_score(self, score):**

**self.score = score**

**Function get\_speed(self):**

**return self.speed**

**Function get\_width(self):**

**return self.width**

**Function get\_height(self):**

**return self.height**

**Function get\_invisibility(self):**

**return self.invisibility**

**Function get\_skin(self):**

**return self.skin**

**Function get\_score(self)**

**Return self.score**

**Skin Unlocking Algorithm:**

**The skin unlocking algorithm checks against the player’s score to see if they have met the requirement to unlock. The unlocked-skin variable is an empty list that will fill up when more skins are unlocked.**

**Function check\_skin\_unlocking(player):**

**unlocked\_skin = [“”]**

**current\_score = player.score**

**# Check if player has reached the required score for each skin**

**if current\_score >= 100:**

**unlocked\_skin = "Skin2.png" # Set skin2 to be unlocked**

**elif current\_score >= 50:**

**unlocked\_skin = "Skin1.png" # Set skin1 to be unlocked**

**return unlocked\_skin**

**Calculating Score Algorithm**

**Function calculating\_score(score, current\_level):**

**score = current\_level \* 5**

**return score**

**Scoreboard Algorithm**

**Function scoreboard(score)**

**playerName = input(“insert playername”)**

**Result = playerName, “:”, score**

**with open(“scoreboard.txt”, “a”) as file**

**file.write(result + ”\n”)**

**Class Diagrams**

**The class diagrams show the various parts of a class, such as properties and methods, in a clear and concise visual manner. They make it simple for developers to comprehend a class's structure and interactions with other classes in the system.**

**Class diagrams can also act as a blueprint for how to implement a class because they outline all required elements and how they relate to one another. This can aid in the development process by ensuring that all essential elements are present and minimising the possibility of mistakes or omissions.**

**The Player class has five private instance variables in this class diagram: speed, width, height, invisibility, and skin. Each of these variables also includes public setter and getter methods, enabling other sections of the code to access and change the player's base stats.**

| **Player** |
| --- |
| **Speed:float**  **Width:int**  **Heightint**  **Invisibility:bool**  **Skin:str**  **x:int**  **y:int** |
| **set\_speed(speed: float)**  **set\_dimensions(width:int, height:int)**  **set\_skin(skin\_name: str)**  **get\_speed():float**  **get\_width():int**  **get\_height():int**  **get\_invisibility():bool**  **get\_skin():str** |

| **Guard** |
| --- |
| **X:int**  **Y:int**  **Width:int**  **Height:int**  **Speed:int**  **Direction:str**  **Range:int**  **width\_range:int** |
| **Move\_left()**  **Move\_right()**  **detect\_player()** |

**Justification**

**The initial task appears overwhelming and challenging, however breaking it down into smaller sub-procedures makes the task a lot simpler to look at and will make the development stage much easier. Also, maintenance and debugging will also be much easier with the aid of modularisation.**

**I have broken down each essential feature into its own class/function. This method of modularising will ensure that all subroutines are interlinked with each other and will be massively useful for finding errors in the code. This means I am less likely to break the code and spend a long time looking for the error.**

**This way of modularising also allows me to make future changes in the near future. Since the code is very modular, development would be as simple as following the design and keeping every feature to its own subroutine.**

**Modularising ensures that i have considered each feature's algorithm and how everything will come together to create a functional end product.**

**Inputs and Outputs**

| **Input** | **Process** | **Output** |
| --- | --- | --- |
| **arrow keys** | **Using pygame’s in-built keyboard input library, i can use keys as inputs** | **Player’s movement** |
| **Buttons(settings/skins/**  **Accessing different levels )** | **Using pygame\_gui, a third party library, i can make more advanced buttons than with the built in pygame feature** | **Will open settings page**  **Will open skins page**  **Will start a new level** |
| **Sliders for:**  **Audio control** | **Using pygame\_gui, I can create a slider for features such as audio control in settings, or scrolling for different levels** | **The value of the audio will increase or decrease depending on the position of the slider.** |

**Key Variables**

**These key variables are what will be used in the program**

| **Name** | **Data Type** | **How it is used** |
| --- | --- | --- |
| **Speed** | **Float** | **Determine how many units the player can travel in one key press** |
| **width** | **integer** | **Determines the width of a player to see if they can fit through a gap or not** |
| **height** | **integer** | **Determines the height of a character** |
| **invisibility** | **boolean** | **Determines whether the player is invisible or not. Can be triggered with power-ups** |
| **Skin** | **string** | **Determines the skin/cosmetic of the player. The image is accessed within a file.** |
| **score** | **integer** | **Determines the score of a player. It is put on a scoreboard and is used to unlock skins** |
| **current\_level** | **integer** | **Used to calculate score and the level transition algorithm** |
| **load\_level** | **boolean** | **Determines whether a level is to be loaded or not, if it is set to True, the next level is loaded** |

**Validation**

**To make sure the program is error-free, and that the inputs are being given the correct data and they won’t output any data when given the wrong inputs.**

**Buttons:**

**Buttons will be provided by pygame\_gui, a third party library in pygame. They will be linked to subroutines that lead to different sections of the game such as the settings page or the levels page. Once pressed, they should go to the desired location.If they are not linked, pressing them will lead to no output.**

**Sliders:**

**Sliders will also be provided by the pygame\_gui library. It will act as a way of controlling audio by moving a slider. The slider will represent a value at certain positions of the bar. The max value of the slider will be 100 and the minimum will be 0. At 0, no audio should be heard from the game and at 100 it should be at it’s loudest.**

**Levels:**

**Levels are a way for players to progress the game and they will gradually increase in difficulty. Levels are locked until the previous level has been completed, hence users will not be able to access the locked levels and pressing them will lead to no output.**

**Skins:**

**Similarly to levels, skins have to be unlocked to be equipped. Skins are unlocked based on the player's score. Once they reach a certain score they will have unlocked the skin. If the player has met the requirements for unlocking a skin, it will appear as unlocked in the customisation page, otherwise, it will be locked and pressing on it will lead to no output.**

**Obstacles:**

**Obstacles are placed around the map to force players to take a different route. If an obstacle is present, the player is unable to walk beyond the obstacle. There should be no way to force a way through an obstacle, the player is forced to take a different route. If the player tries to walk beyond the obstacle, there will be no output.**

**Power-ups:**

**When a player walks over a power-up, they equip it for the rest of the game. A level may require the user to use the power-up otherwise it will not be possible to beat the level. This means that a player must walk over a power-up, if they don’t they should not have the power-up. If they walk over it, it should have an effect on their base stats.**

**Guard Detection:**

**Guard’s detection range is a rectangle with width and height. If the player has stepped inside the range, the game will be ended as the player has failed the level and been caught by the guard.**

**Testing**

**Throughout the development stage, testing shall be done on each modular component of the game to test if it works as intended. All inputs done shall be recorded along with the outputs. This method of testing will ensure that playing the game as intended will have very few bugs.**

**After the development stage, a final test will be done again by me to perform destructive testing. This will be done to find as many bugs as possible and test the game’s limits. This is essential as it will make the game’s components robust and almost error free.**

**All inputs will be recorded into a separate table for destructive testing along with the outputs. This will allow me to debug the game for a final time before releasing a prototype to my stakeholders to see if it is to their liking.**

**Iterative Development**

**I have decided to use Iterative development as it is a method that involves creating a basic version of a product, testing it, receiving feedback, and then refining it over the course of several iterations. Once the product is finished and meets the required requirements, the procedure is repeated.**

**This approach also permits ongoing feedback from users and stakeholders, which can assist guarantee that the final product satisfies their requirements and expectations. By identifying issues early on and taking immediate action to fix them, it can also lower the possibility of schedule delays.**

**Overall, iterative development is the most appealing method to me as it allows me to create an end product that is suited to the liking of my stakeholders with low risk of having to redo a lengthy section of development.**