L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

Drained
Game Jam 2022 (Group 2)

Gabriel Borg* (0235902L), Matthias Bartolo** (0436103L), Jan Lawrence Formosa** (0435502L), Andrea Avona*** (0227624A), Anthony Mifsud* (012002L), Jonathan Polidano*** (0191003L)

*B.Sc. It (Hons) Artificial Intelligence (Third Year)
**B.Sc. It (Hons) Artificial Intelligence (Second Year)
***B.Sc. It (Hons) Software Development (Second Year)

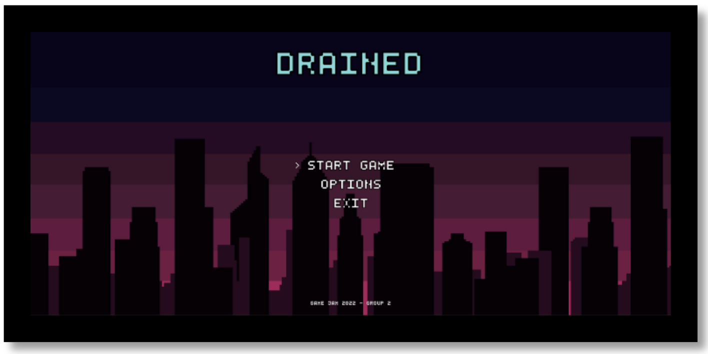
Study-unit: **Game AI & Advanced Game AI**
Code: **ICS2211 & ICS3209**
Lecturer: **Prof. Alexiei Dingli**

Table of Contents

Conceptual Documentation of the Game

General overview of the game

Drained was created as part of the 2022 University of Malta Game Jam and is a side-scrolling platformer game. The gameplay revolves around exploring the play area to find collectible items intended to relieve mental health exhaustion, while avoiding obstacles. The game also adheres to the theme of the Game Jam, which was **Mental Health**, and features several AI-related features which were covered in both the Game AI and Advanced Game AI courses. Moreover, an effort was made to render the game fun, whilst exhibiting characteristics of challenging gameplay, a compelling endless mode, and complementary game mechanics. The visual images of the introductory page and gameplay are illustrated by Figures 1 and 2 respectively.



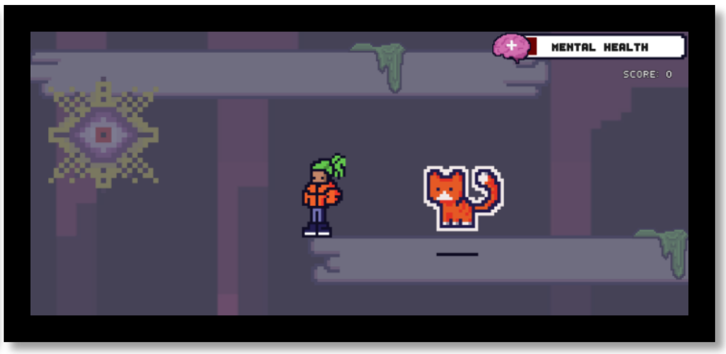


Fig 1 and Fig 2.: The Main Menu of Drained, followed by a screenshot of a part of the gameplay.

Application of the theme

The team took great care in making sure that Drained accurately represents the theme of mental health. This is reflected through the player who, in the game, takes control of a character burdened with the effects of mental exhaustion. The primary representation of this is a mental health bar visible in the top-right corner of the screen, which goes down over time, thus simulating being in a "draining" situation. Like real life mental health afflictions, the bar going down does not harm you physically, but negatively impacts you in other ways. When one is mentally exhausted, common symptoms are reduction in mental clarity and scattered thoughts, as mentioned in [1]. In the game, this is demonstrated by the visuals becoming fogged up, reducing the player's ability to decipher what is on screen.

Another element directly related to the mental health bar are the relaxation spots collected in the first level of the game. These spots represent recognised ways to treat mental stress, such as exercising or eating healthy [2]. The idea is that these will help the player to partially restore their mental health status. However, these also become less effective when used repeatedly, to show that the character is becoming desensitised to these methods over time.

The general presentation of the game's elements is also designed to fit into the theme. For example, the enemy which chases the player is not a representation of mental fatigue itself, to avoid demonising it, but rather a vaguer figure called "the Eye" which refers to the judgmental eye of society. The stationary hazards cause the player to lose the game, as they are now "drained". In fact, whenever the player loses, they have to start over, due to them feeling exhausted.

The first level of the game takes place in a cityscape scenario, with the user jumping on floating clouds which are platforms used to traverse throughout the map. The second level uses the same platform representation; however, it takes place in a dreamscape. Consequently, the mental health bar in the first level is now replaced with useful mental health tips for the player. In this level, the player is still surrounded by threats represented by “the Eye”, which are now multiplied. There are now three of them, with the power to phase through walls, thus being more dangerous. This mechanic was implemented with the aim of truly representing “the Eye” as an imaginative entity, with the unrealistic enhanced ability of going through clouds. Despite the unrealistic setting we still wanted to show that a real threat to the player’s wellbeing is evident.

Outline of game mechanics

Player Movement

Player movement is one of the most fundamental mechanics of video games. It comes in different forms, from a simple point and click mechanic to the most complex movement systems for 3D games. The player movement in Drained, inspired by [3], is simple but effective. A rigid body is pre-emptively attached to the player object in unity to control the movement through code and the player pressing keys. Since Drained is a 2D game, we are concerned only by the player movement in the horizontal axis and in the vertical axis. The player’s array of movements is clearly seen in Figures 3 and 4.

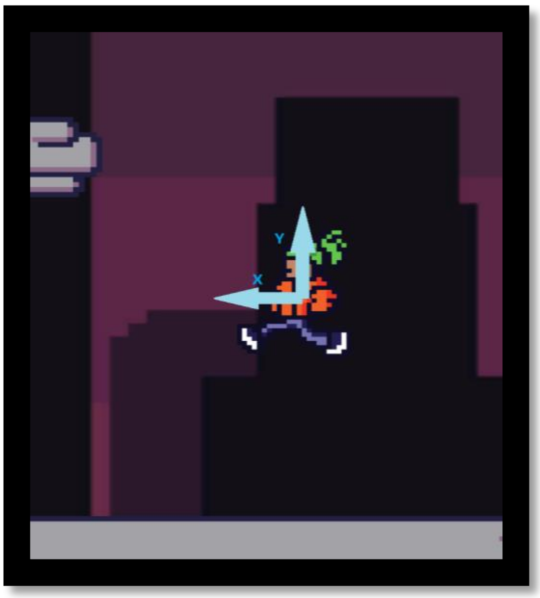


Fig. 3: Player jumping left

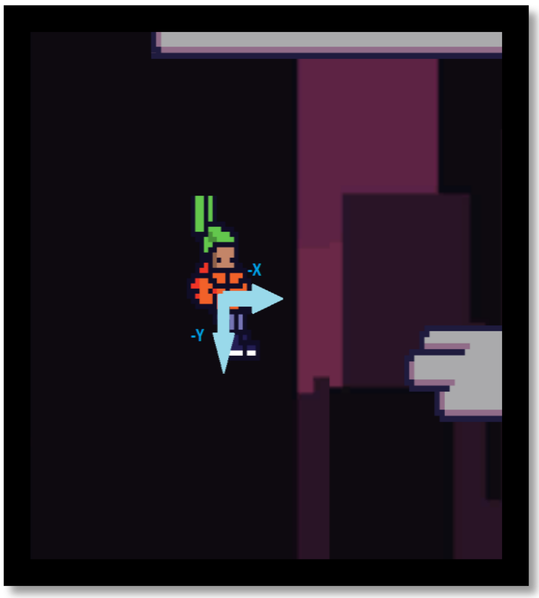


Fig. 4: Player falling right

As mentioned before, in the PlayerMovement.cs script, we get the Rigidbody2D [4] component in the Start function. In the Update function, we first get the direction the player is moving in horizontally by checking if the player is pressing either the left/right arrow keys or the keys *A/D*, then we multiply it by the velocity (determined before in the code) and apply it to the rigid body to push it horizontally. The key that the user would press is obtained using the Unity standard Input Manager [5]. Next, the vertical component is determined by the jumping of the character. The force is applied to the rigid body only if two conditions are met: the player presses the *spacebar* or the *W* key on the keyboard, and the character is “grounded”. This is done to prevent the player from jumping while in mid-air.

It is pertinent to take note of the direction, the character is facing while moving. Attention was paid to this in the script controlling the player movement, enabling a pre check of the player’s direction of motion. This ensures that there is the necessary change in the sprite’s direction, which will make sure that the player faces the right direction. The same mechanism was implemented for the jumping animation and falling animation.

The avoidance of collisions with the platforms in the game was another detail which was addressed. The issue, faced by the team entailed that, whenever the character collided with the side of a platform, and the key for directional movement was kept pressed, the character got stuck mid-air on the side of the platform. This is due to the rigid body default material, which had friction. However, by simply introducing a physics material with no friction, the character simply falls off the platforms, whilst avoiding getting stuck on their sides.

Collectibles

Player engagement can be regarded as a critical component in a game’s dynamic. In Drained, such a concept was implemented through the introduction of a collectibles reward system. Drained includes collectibles which vary depending on the different levels of the game. In the first stage, these are depicted as relaxation spots, whereas in the second stage, instead of relaxation spots, the player needs to collect the tasks collectibles. All of the collectables are prefabs spawned in the game after the scene and map were loaded. They were assigned a box collider component to check when the character’s collider and the collectible’s collider intersected [6]. In the event that this collision occurs, there is a deactivation of the collectibles’ game object, while simultaneously playing a sound. The spawning of relaxation spots and tasks are done in a pre-determined manner, which ensures that the first generation of spawn points have a different range of effects from the next ones. Further elaboration on this mechanism will feature in the section titled AI features.

Relaxation Spots

In the first stage of the game, the player must collect "relaxation spots”. These spots keep the mental health bar from draining whilst also depicting everyday scenarios, where some minor actions can truly help people relieve stress. These small actions might be in the form of eating a healthy meal, socialising, or exercising which all help to maintain a healthy mental lifestyle. Some of the aforementioned actions are reflected in Figures 5, 6 and 7. In Drained, these come as collectibles that the player must look for. Once they are collected, the mental health bar will be replenished by a certain amount to symbolise that the character is feeling less stressed. Once the player collects sever relaxation spots, they will proceed to the next stage of the game, unless they are playing in endless mode. In endless mode, the player will continue playing in the first stage until colliding with a trap or enemy with the objective being to obtain a high score.

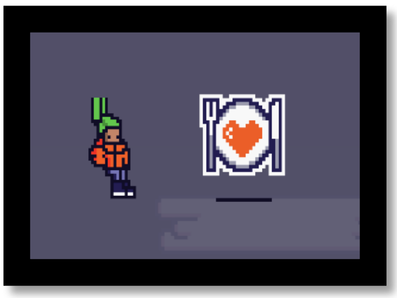


Fig. 5: RP meal

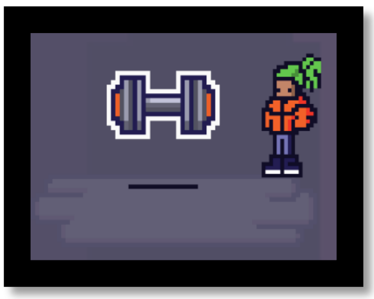


Fig. 6: RP exercise

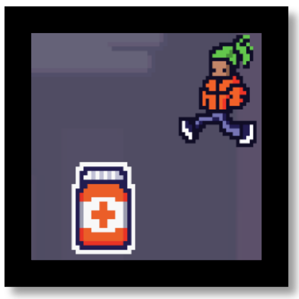


Fig. 7: RP medications

Tasks

In the final stage of the game, the player need not concern themselves with the mental health bar. The objective has become to collect enough tasks, which symbolises the character rolling up their sleeves and getting stuff done. This is often an even more stressful time for people. To symbolise this added pressure, the number of enemies chasing the player was increased to three. The tasks in the game are represented by glowing stars, as shown in the Figure 8.

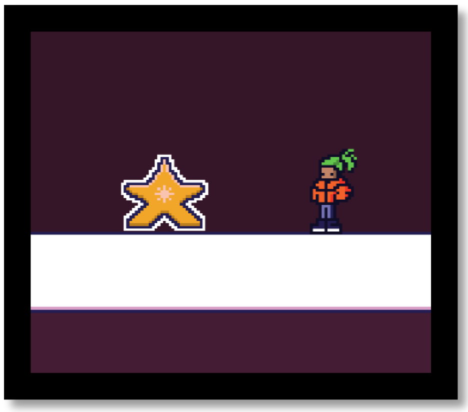


Fig. 8: Collecting the star

Screen Blur Effect and Mental Health Bar

As the game revolves around the theme of mental health, one of the most critical game mechanics is the mental health bar. This bar acts as a meter depicting the player's mental state. A full mental health bar is an indication that the player is energetic, enthusiastic and a truly healthy person at peak mental fitness. However, as with real life, stressful events tend to negatively affect the mental health of a person. To mimic such behaviour, the mental health bar in Drained decreases with time. Fortunately, this mental health bar can be replenished, through the seeking and collecting of the previously mentioned relaxation spots. Finally, the mental health bar is also responsible for spawning the relaxation spots and only does so upon occupying 2.5 times the bar's volume. This allows the player to replenish it using the spawned relaxation spots.

An effect that was added to the game as an immersive game mechanic is the screen blur effect, which is directly linked to the mental health bar. The idea was that, as mental health slowly deteriorates with time, the screen starts getting blurry. This was added to simulate the accumulation of mental fatigue, whilst presenting a visual depiction of the player slowly being "drained". Eventually this blur produces a whiteout on the screen, compromising the player's vision and making it harder for them to see traps. The whiteout concept will be clarified in the coming sections.

Traps

In the first stage of the game, traps are spread out across the platforms. If the player touches one, they will instantly get drained, resulting in a game over. While increasing the difficulty this also symbolises how sometimes small, unexpected things could have a great backlash. In the game, the traps are shown as green slime, seen in Figures 9 and 10. The designs of the slimes was chosen to complement the city aesthetic. They were made from prefabs game objects that had two simple components attached to them, a Sprite Renderer [7] and a Box Collider2D [6]. Then they were tagged as "Trap" objects. Finally, we can see in the PlayerLife.cs script that when the player's collider intersects the collider of any trap, it will call the "Die()" method. This is shown with the player becoming drained of colour in Figure 10. This will result in a game over for the player and a consequent switch to the game over scene.

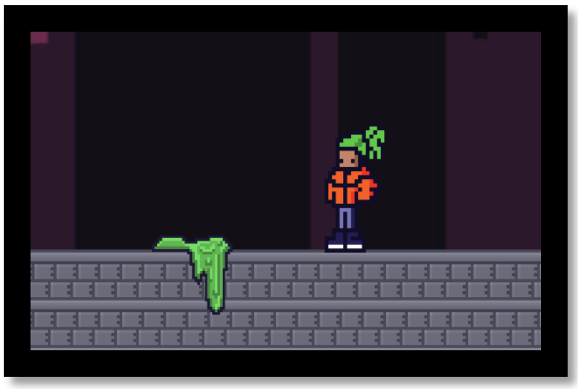


Fig. 9: Traps

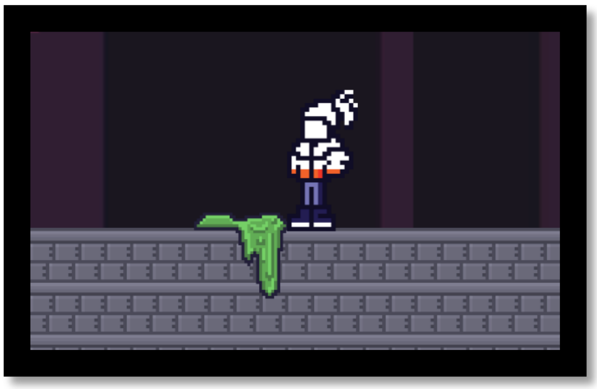


Fig. 10: Player draining after touching a trap

UI

The Start Menu is the first thing that the user comes across upon opening the game for the first time. Here, the user can navigate either with the arrow keys or the mouse to either the "Start Game" section, the "Options" Section, or the "Exit" button, which closes the game. The arrow key navigation mechanic was added for two reasons. Firstly, to simulate the feel of the Directional Pad on a controller, as analogue sticks were not yet part of playing older games such as the original Sonic the Hedgehog [8]. Secondly, the option to use the arrow keys to navigate was added to aid left-handed players, who typically use the mouse with their left hand and keyboard with their right hand while gaming. One can also press the "enter" key or the "space" key to confirm their choice.

Upon clicking the "Start Game" button, the user will be presented with a choice of two game modes: "Normal" Mode and "Endless" Mode, as can be seen in Figure 11. The Normal game mode, upon being selected, will spawn the player in the first level, where the user must collect 7 relaxation spots to progress in the game. On the other hand, the Endless game mode will spawn the player in a different version of the first level, where the goal for the user is to survive for as long as possible and gather as many points as possible. It must be noted that whereas in the Normal mode, the player can go from the first level to the second level, in Endless mode, the player remains in level one, to collect as many relaxation points, as the player so pleases. This can be shown in Figure 12.

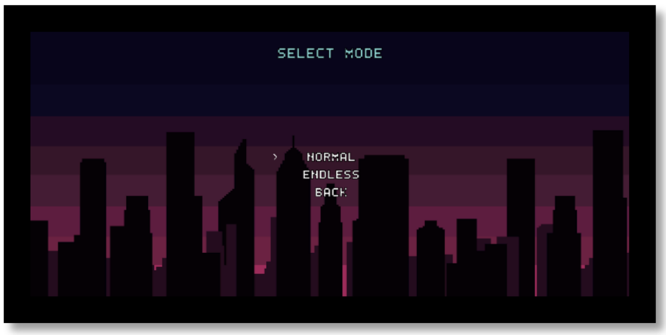


Fig. 11: Screenshot showing the options for the user to pick either the "Normal" or "Endless" game mode.

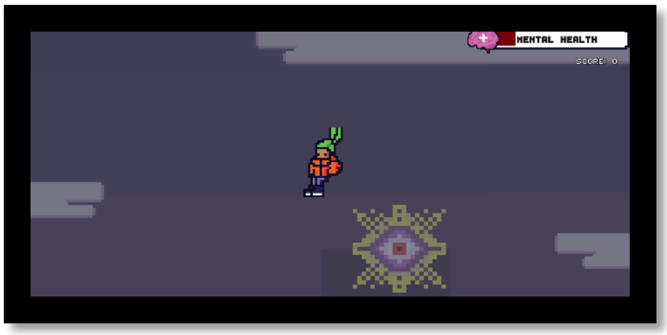


Fig. 12: Gameplay image of the first level of the "Normal" and the "Endless" game mode.

In the start menu, the user can also navigate to the "How to Play Section" through the "Options" section, as clearly exhibited in Figure 13. This section shows a panel with controls and imagery to teach the player not only about the basic navigation controls, but also useful information about what the traps look like, the enemy and the mental health bar. This allows the players to grasp a basic understanding of the game before they start playing it, rather than being thrown into the game without prior knowledge on how to play.

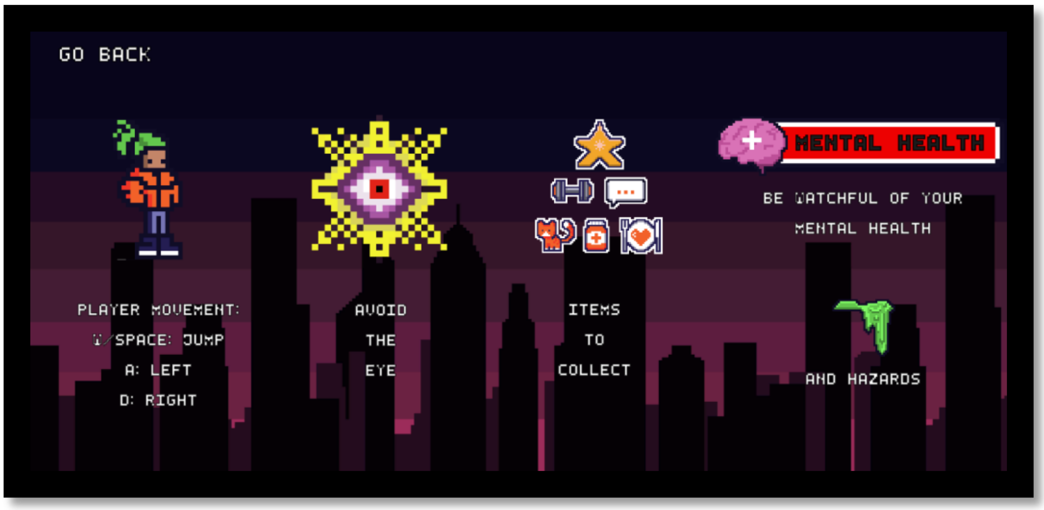
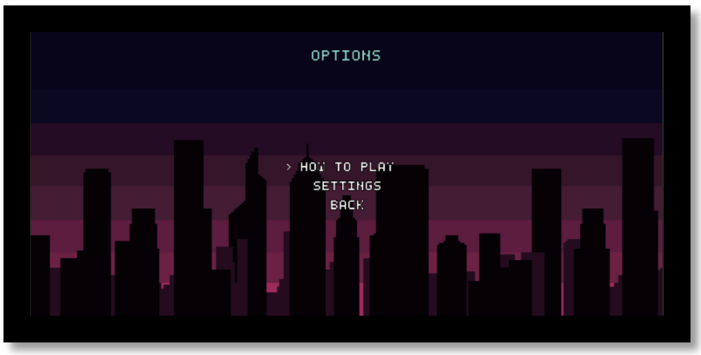


Fig. 13: Image showing the "How To Play" section.

The second option button being the "settings" button, allows the player to change the game's Music and SFX sounds through the respective sliders. Continuously, the player, would be able to set the specific sound component to the desired sound volume individually, as demonstrated in Figure 15.



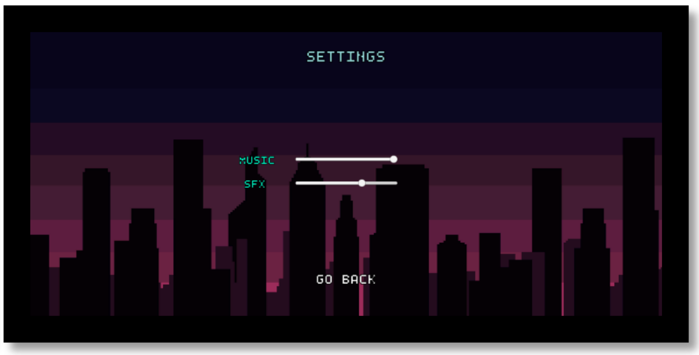


Fig. 14 and Fig. 15: Images showing the “Options'' and “Settings” section.

When the player manages to beat the game (complete both stages), the victory menu will appear. Here the game will congratulate the player and their score will be shown as exhibited in Figure 16. The player will then have the option to have another go at the game or to go back to the main menu.

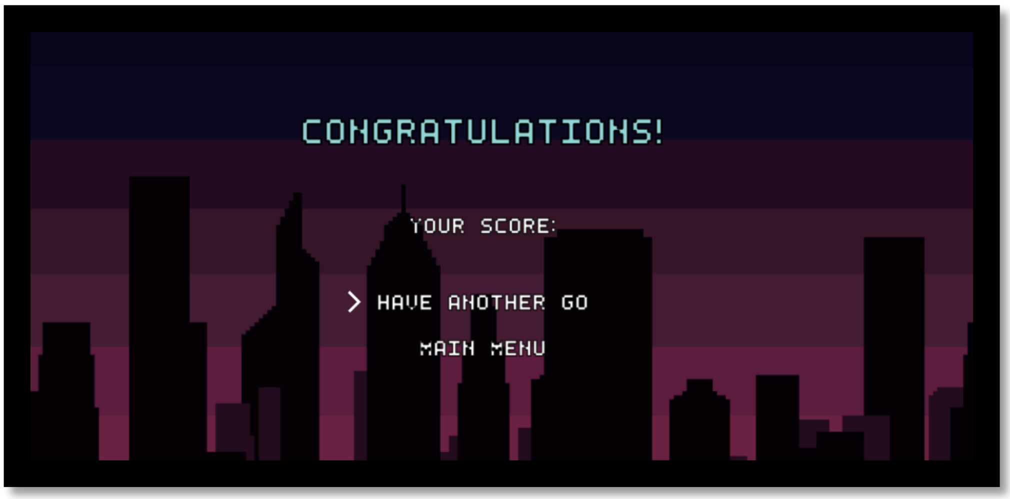


Fig. 16: Image showing the “Congratulations” end screen.

Once the player gets drained, the game over screen is presented. A menu will appear giving the player the options to either play again, enticing them to have another go, or to be redirected back to the main menu, as demonstrated in Figure 17. The behaviour of the buttons is analogous to the previous screens.

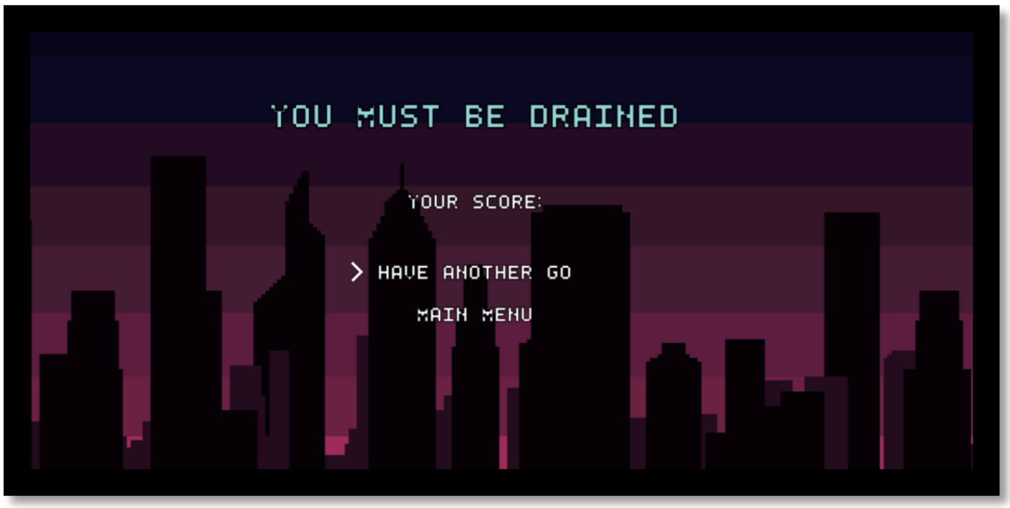
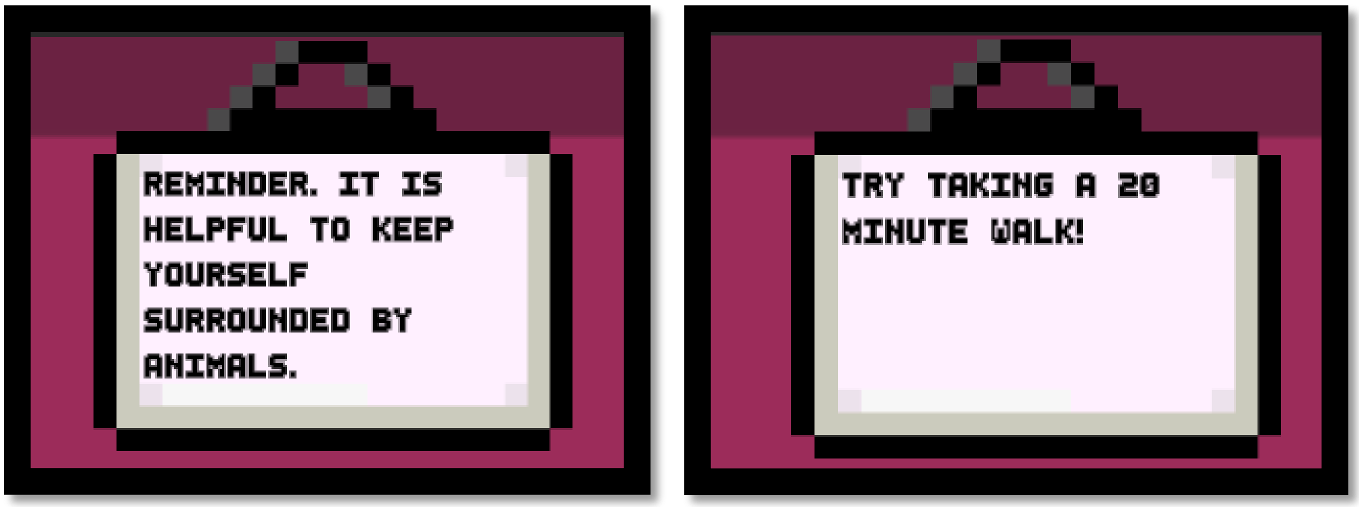


Fig. 17: Image showing the "Drained" end screen.

To continue to stick with the theme of mental health, another UI element was designed within the game, this being in the form of mental health tips. These tips include educational advice which appear inside a noticeboard image situated at the top right-hand corner of the screen. The advice shown to the user, which changes periodically, tackles topics intended to improve one's mental health. These tips focus on the fundamentals of following a balanced diet, the need of exercising for mental strongness, the positivity of enjoying animals' company, and the usefulness of surrounding oneself with trustworthy people. A visual example, of some of these tips can be seen in Figures 18 and 19 respectively.

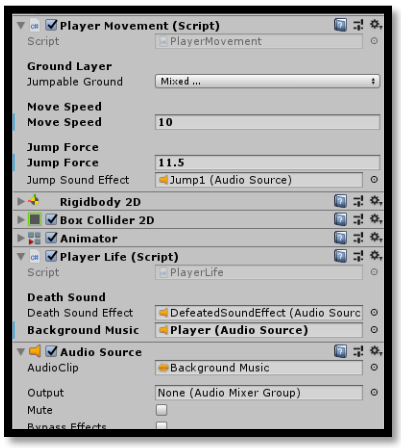

REMINDER. IT IS HELPFUL TO KEEP YOURSELF SURROUNDED BY ANIMALS.

TRY TAKING A 20 MINUTE WALK!

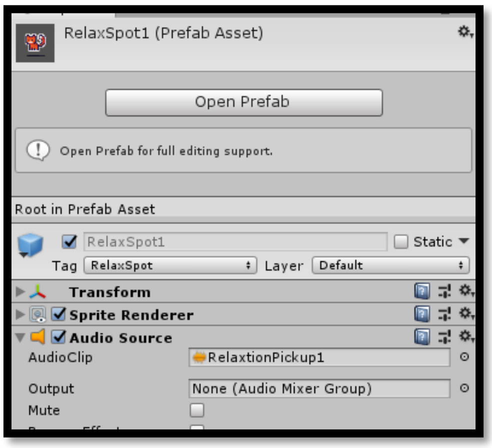
Figs. 18 and 19: Images showing different mental health tips presented to the user.

Sound

Although, sometimes overlooked, sound possesses a key element in game development. In Drained, the sound was made entirely by us (except for a sound effect [9] used when the player loses), using a combination of tools. The app BandLab [10] was used for the music and the sound effects tool SFXR [11] was used to make the sounds for jumping, collection of items, replenishing the meter, etc. Audio can be adjusted in the main menu by two distinct sliders for the background music and the sound effects respectively. Some inspiration for the sound system was taken from [12], which helped to enable the creation of our sound. These sounds are played by different audio sources attached to different game objects. For instance, the player is the one producing the jumping sound (when they jump), whilst the background sounds like the defeat sound and the music are continuous. These sounds are controlled via a script, namely PlayerLife.cs and PlayerMovement.cs. These scripts include audio source variables that are then added via the editor. These variables allow us to control what sounds plays when the W/Space bar key is pressed and what sounds plays when the player has died and so on. Other objects tasked with emitting sound are the collectibles. All of them have an Audio Source component attached to them, the relative audio clip is then played every time a collectible is touched by the player, and hence picked up. This is again controlled by a script. This can be seen in Figures 20 and 21.



```
Player Movement (Script)
Script                PlayerMovement

Ground Layer
Jumpable Ground       Mixed ...

Move Speed
Move Speed            10

Jump Force
Jump Force            11.5
Jump Sound Effect     Jump1 (Audio Source)
Rigidbody 2D
Box Collider 2D
Animator
Player Life (Script)
Script                PlayerLife

Death Sound
Death Sound Effect    DefeatedSoundEffect (Audio Sourc
Background Music      Player (Audio Source)
Audio Source
AudioClip             Background Music

Output                None (Audio Mixer Group)
Mute
```



```
RelaxSpot1 (Prefab Asset)

Open Prefab

Open Prefab for full editing support.

Root in Prefab Asset
RelaxSpot1                          Static
Tag  RelaxSpot           Layer  Default
Transform
Sprite Renderer
Audio Source
AudioClip             RelaxtionPickup1

Output                None (Audio Mixer Group)
Mute
```

Figs. 20 and 21: The audio sources for both the background music and the relaxation spots

The volume was kept consistent between the scenes of the game. This is done with the class "PlayerPrefs" [13] which keeps the set values of the sounds stored, even when switching scenes or closing the game. This class was used in combination with the game objects "AudioManager" and "AudioSetter", which were linked with their respective scripts. We can find the first object in the main menu-and the second one throughout all the scenes of the game. With the AudioManager.cs script, we perform a check to determine whether the player has already set any values for the volume before so that the software will automatically reactivate such settings thus saving the player time. In this script, we also find the UpdateSound() method, which keeps the volume consistent with the values from the two sliders. This system was created following Greg Eads' tutorial [14].

AI Features

Genetic Algorithms

Genetic Algorithms (GAs) are algorithms which simulate the process of natural selection [15]. This refers to those species who can adapt to changes in their environment and are able to not only survive but reproduce and go to the next generation in a "*survival of the fittest*" based scenario. The GA contains several **generations** for solving a problem. Each generation consists of several **individuals** called a **population** and each individual represents a point in the problem search space and possible solution. Each individual is represented with a **chromosome**, which is as a sequence of data, whether that be a string of characters, integers, floats, or bits.

A Fitness Score is given to each individual which shows the ability of an individual to mate. The GA assigns a fitness score to each individual, and the probability that an individual will be selected for reproduction is based on the fitness score that is given to it. The GA maintains the population of *n* individuals (chromosome/solutions) along with each of their fitness scores. Those individuals having better fitness scores than other individuals in the space will be given a better probability to mate and are more likely to get chosen for this process. Those individuals are then selected as pairs of parents to mate and produce offspring by combining chromosomes of parents. This process is repeated until convergence, whereby the GA would keep on mating pairs of individuals to produce children having better chromosomes until a possible solution is found to the problem.

Selection, Mutation and Crossover

The Selection Process refers to the GA giving preference to the individuals with good fitness scores, whilst allowing them to mate, and create a better population.

The Crossover Process represents the mating between individuals in specific crossover sites (chosen randomly) after the selection process finishes. In this process, the genes of the selected individuals at these crossover sites are exchanged, creating a completely new individual, which would represent the offspring.

The Mutation Process implies the insertion of random genes in offspring to maintain the diversity in the population to avoid premature convergence of the algorithm.

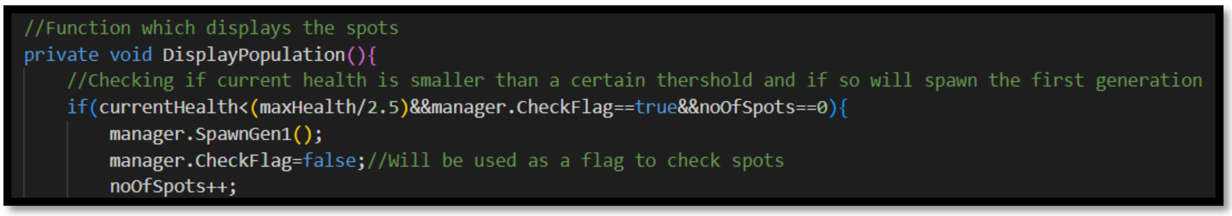
Genetic Algorithm Basic Pseudocode [16]:

```
1) Randomly initialise population $p$
2) Determine fitness of population
3) Until convergence repeat:
    a) Select parents from population
    b) Crossover and generate new population
    c) Perform mutation on new population
    d) Calculate fitness for new population
```

The algorithm converges if the population does not produce any more offspring which is better than the previous generated generation. It is then said that the GA has generated a solution to the current problem.

Mental health bar

The way genetic Algorithms were used in Drained was heavily inspired from [17] and connects with the usage of relaxation spots and their ability to replenish the user's mental health bar. When a user is traversing the map and the mental health bar gets below 2.5 times its length, the relaxation spots spawn around the map. This is displayed in Figure 22.

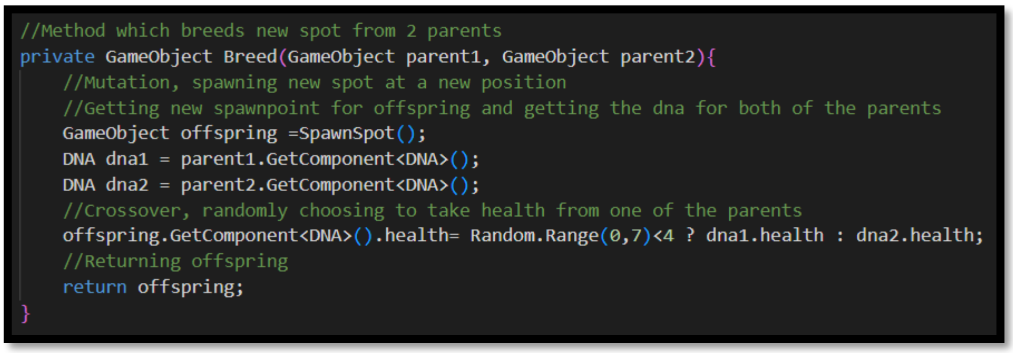


```
//Function which displays the spots
private void DisplayPopulation(){
    //Checking if current health is smaller than a certain thershold and if so will spawn the first generation
    if(currentHealth<(maxHealth/2.5)&&manager.CheckFlag==true&&noOfSpots==0){
        manager.SpawnGen1();
        manager.CheckFlag=false;//Will be used as a flag to check spots
        noOfSpots++;
```

Fig. 22: Snippet of code showing the trigger for spawning the first generation of relaxation spots upon the mental health bar reaching 2.5 times its length.

The user can then collect these spots to fill up their mental health bar with a certain refill value. This value will decrease over time, to represent the slow decline of mental health. However, upon collecting of the spots the mental health bar will be replenished. As one advances the game will continuously get harder and harder, needing more relaxation spots for the bar to be replenished. The reason why this AI technique was added was to simulate the effect of mental exhaustion. As time goes on, the relaxation spots become less and less effective, simulating how over time the player will get more and more drained of energy, as the screen slowly fades to white.

The first generation of spots, with a population size of **5** are spawned into the map. Each spot is generated randomly based on a range and from a pool of spawn points. Once the player touches a spot, it becomes disabled. Touching all the spots will breed the next generation as shown in Figure 23. The new bred generation along with each spot's health replenishment counter converges to a local minimum, and all spots will eventually have the same value. **Mutation** takes place when spawning a new relaxation spot and obtaining the new spawn point. The **Crossover** process occurs by randomly choosing the health taken from one of the parents through a randomly generated index. Such health will be part of the offspring's DNA make-up.



```
//Method which breeds new spot from 2 parents
private GameObject Breed(GameObject parent1, GameObject parent2){
    //Mutation, spawning new spot at a new position
    //Getting new spawnpoint for offspring and getting the dna for both of the parents
    GameObject offspring =SpawnSpot();
    DNA dna1 = parent1.GetComponent<DNA>();
    DNA dna2 = parent2.GetComponent<DNA>();
    //Crossover, randomly choosing to take health from one of the parents
    offspring.GetComponent<DNA>().health= Random.Range(0,7)<4 ? dna1.health : dna2.health;
    //Returning offspring
    return offspring;
}
```

Fig. 23: Function which calls for the breeding of the new population, found in the *Population Manager* script.

This means that as the player keeps on making contact with these relaxation spots, the value that replenishes the health bar will decrease, enabling the team to create the "Endless" mode in the game apart from the "Normal" mode. Figure 24 illustrates the spawning of the relaxation spots in the map.

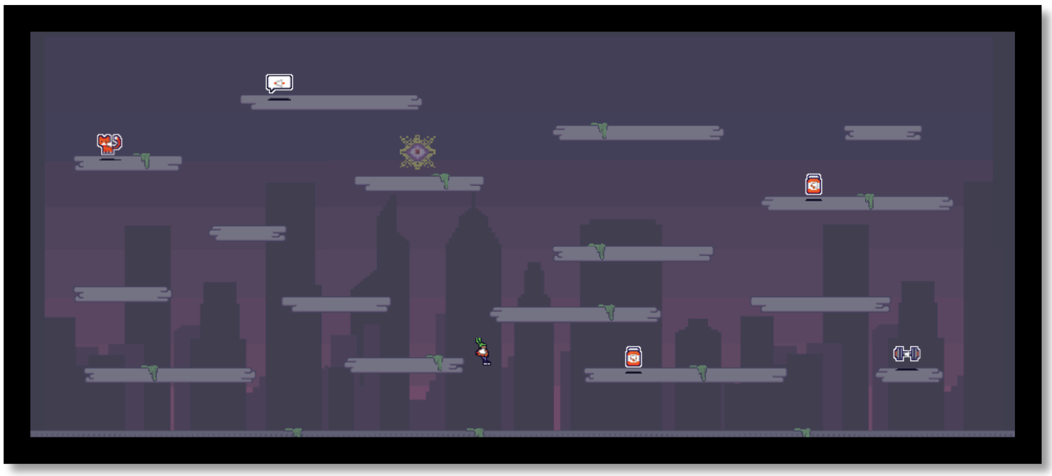
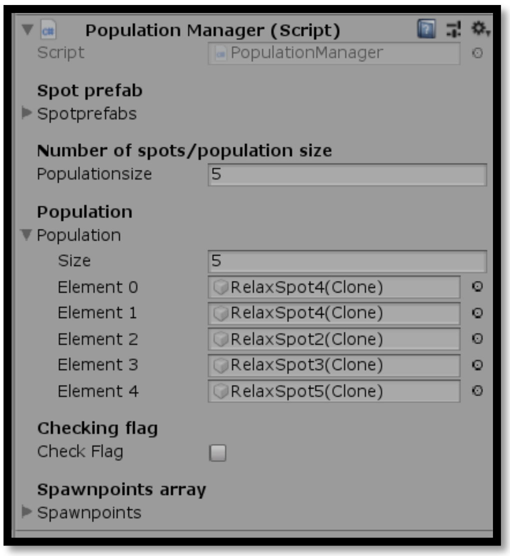


Fig. 24: Showing the spawning of the relaxation spots in-game.



Population Manager (Script)

Script	PopulationManager

Spot prefab

▶ Spotprefabs

Number of spots/population size

Populationsize	5

Population

▼ Population

Size	5
Element 0	RelaxSpot4(Clone)
Element 1	RelaxSpot4(Clone)
Element 2	RelaxSpot2(Clone)
Element 3	RelaxSpot3(Clone)
Element 4	RelaxSpot5(Clone)

Checking flag

Check Flag	☐

Spawnpoints array

▶ Spawnpoints

Fig. 25: The *Population Manager* script handles the randomly generated spots for the relaxation spots to be spawned in, as well as showing the population size.

One final note to add is that since the locations where the relaxation spots will be spawned are chosen at random, this implies that **there is no specific pattern that the user can get familiar with**. This element of randomness was created not only for the purpose of <u>scalability</u>, whereby there is an option to increase the population size, but also to <u>provide a challenge</u> for the player and keep them alert, thus forcing them to traverse the entire map and not just specific portions of it. Figure 25 illustrates the Population Manager.

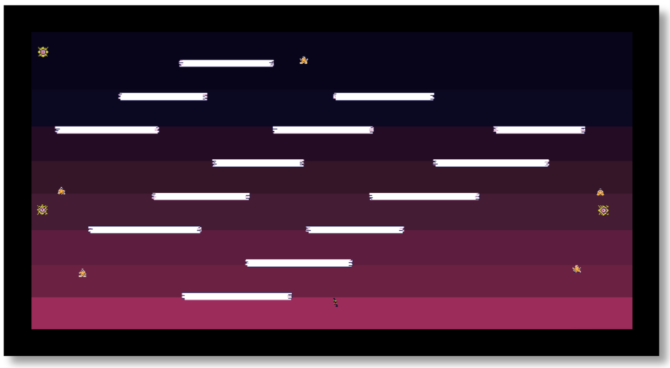
Procedural Map Generation

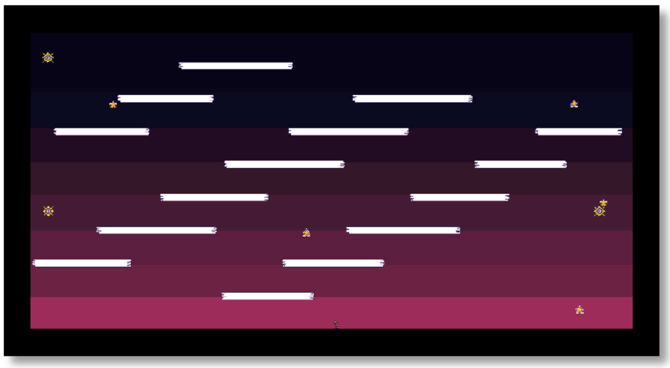
As stated in [18], Procedural Map Generation is described as "*the ability to create partially random content by the computer*". This means that by providing the program with some element of basic templates, terrain, or assets and mixing it with a procedural generation algorithm, one could now spawn an infinite number of maps and worlds to be played by the user. This element of modularity was added to increase playability within the game and reduce redundancy in game assets. Procedurally Generated content can be spawned in the beginning of every playthrough, adding new challenges to the game, furthermore keeping the player on alert since new environments bring new challenges.

Drained was designed with three methods for procedurally generating the map which the player is immersed in. These are structured on the Depth First Search platform generation technique, a Breadth First Search platform generation technique, and a Recursive platform generation technique. These methods are assigned an index. When the player presses "Play", a random number is generated, and a map is chosen with the index corresponding to that random number chosen. This implementation allows for the possibility that users play the game in either a DFS, BFS or Recursively populated environment.

DFS

The Depth First Search platform generation algorithm works by **visiting all the nodes down one path until there are no more nodes to traverse** [19], it will then backtrack until it reaches a node which can be traversed. This process repeats itself until the tree is completely explored. Moreover, DFS begins at the root node and proceeds through the nodes as far as possible until we reach the node where there are no unvisited nearby nodes. The population method for filling in the terrain for one of the maps consists of loading platforms in a pyramid shape. The algorithm *SetPlatform1* used in the script *ProceduralMapFinal* implies that when the platforms spawn, they are spawned until there are no more places to spawn the tiles for the platforms. This ultimately results in a map with several layers of platforms in a repeated pattern, and the user can jump across the different levels. These are clearly shown in Figures 26 and 27.



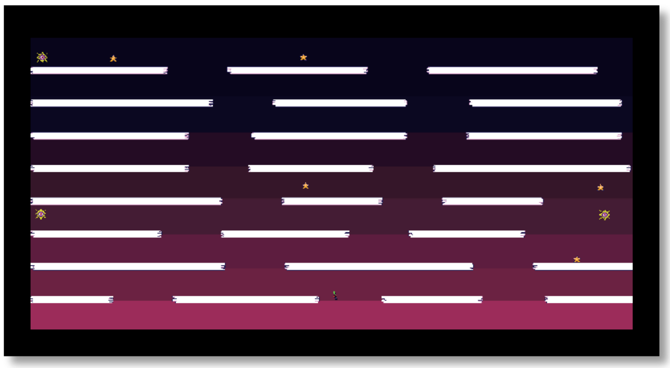
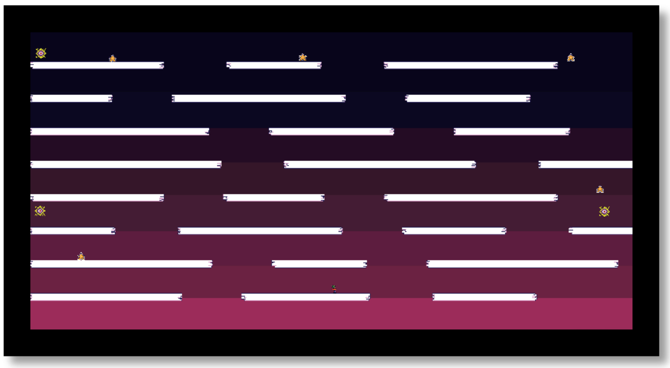


Figs. 26 and 27: Figures showing the first map possibility, being the Depth First Search population. One can notice the pyramid pattern of platforms, stretching down the map until there is no more space to spawn.

BFS

The Breadth-First-Search platform generation algorithm differs a lot from DFS since the fundamental structure does not use a stack [19]. Unlike DFS, it uses a queue. This traversal consists of **going through**

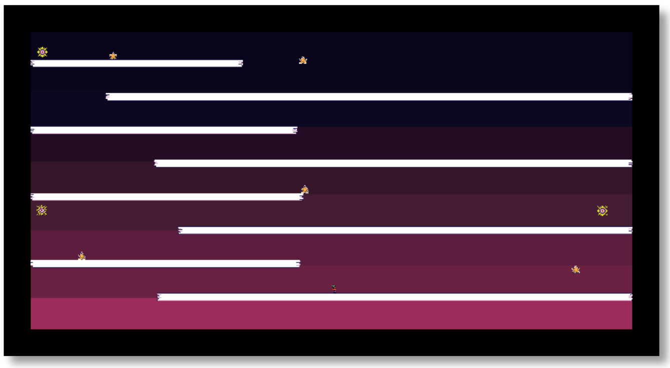
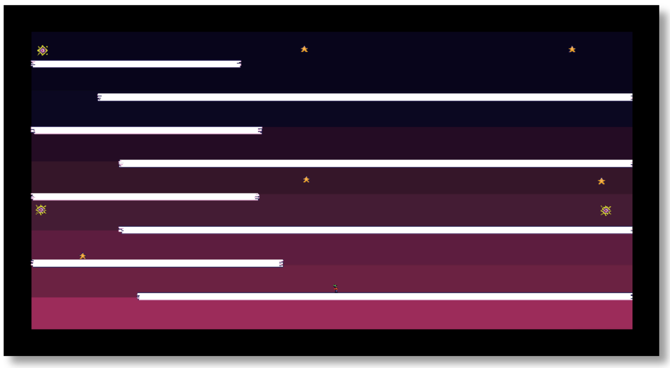
all nodes on the same level before moving on to the next level. The algorithm *SetPlatform2* found in the same script spawns the tiles for the platforms one row at a time while leaving a small gap, bigger than the size of the player, going down one level when there is no more space to spawn another platform. This results in several platform lines which the player can run on, and upon comparing to the DFS map population methods, this **increases the number of platforms to spawn** since it is more space efficient when populating the map. This is being demonstrated in Figures 28 and 29.

Figs. 28 and 29: Figures showing the second map possibility, being the Breadth First Search population. One can notice the increased number of tiles, and the uniform lines of platforms spanning across and down the map.

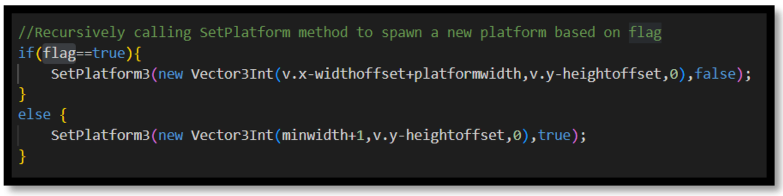
Recursion

The third and final method for generating platforms in the map is through another recursive algorithm, like DFS. This algorithm, named as *SetPlatfrom3* and is again found in the same script, works by spawning a tile map, and then recursively calling the same function again to spawn another platform based on a calculated height and width offset, platform width and a check flag. Conclusively, Figures 30 and 31 depict this generation process.

Figs. 30 and 31: Figures showing the third map possibility, being the recursive generation method.

The result is a different map layout, where the platforms are placed in a staggered formation, from opposite sides of the previous platform. Each platform also has different lengths, decreasing uniformity. Figure 32 illustrates the code used.



```
//Recursively calling SetPlatform method to spawn a new platform based on flag
if(flag==true){
    SetPlatform3(new Vector3Int(v.x-widthoffset+platformwidth,v.y-heightoffset,0),false);
}
else {
    SetPlatform3(new Vector3Int(minwidth+1,v.y-heightoffset,0),true);
}
```

Fig. 32: A snippet of code showing the recursive call to spawn another platform. This is located at the very end of the function body, thereby re-calling the function again until a stop flag terminates the creation of more platforms.

Enemy AI (A* Pathfinding)

The A* algorithm [20] was used for enemy pathfinding in the game where the platforms served as obstacles. The first step was to scan the map and identify all the nodes, which were the edges of the platforms. Then, the enemy used A* to try to find the player, while also avoiding obstacles. Once the path was found, the enemy follows it towards the player until either the player completes the level, the player hits a hazard, or the player is caught by the enemy. To follow the path, the enemy determined the direction of the next waypoint and applied a force in that direction until reaching the waypoint, at which point it would have a new waypoint to follow.

A* is like Dijkstra's algorithm, but it is more efficient because it avoids exploring nodes that are too far from the endpoint when compared to other, more promising nodes. The cost of each node in A* is calculated using the formula $F = G + H$, where F is the total estimated cost of the current node, G is the cost from the starting point to the current node, and H is the estimated cost from the current node to the end point. Additionally, for each node, the parent node must be kept track of for backtracking purposes.

At the beginning of the algorithm, the F, G, and H scores for all nodes are set to infinity since the algorithm has no information about them yet. The algorithm starts by popping the node with the lowest F score from the open list, which initially consists of just the starting node. When a node is popped from the open list, its neighbours are visited, and their F scores are calculated. Whenever a node is visited, its scores are updated to keep the smallest values between them, and the previously visited node is replaced if the current path is the smallest one so far. The algorithm finishes when the final node is popped from the open list, at which point the algorithm must backtrack to the starting point to determine the complete shortest path. Figure 33 illustrates the pathfinding algorithm.

Implementation of such an algorithm was facilitated through [21].

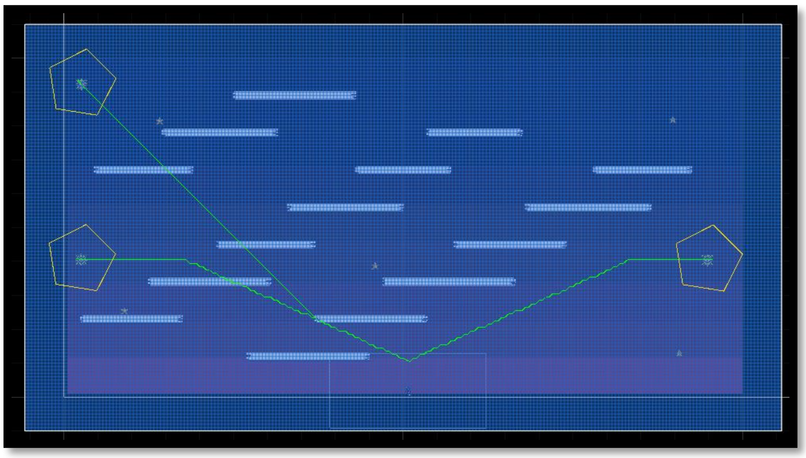


Fig. 33: Screenshots showing the pathfinding algorithm for all three enemies. One can also note the path calculated and highlighted.

Finite State Machines.

Finite State Machines (FSMs) are a useful tool for representing the behaviour of characters in video games, particularly when the character has a limited set of possible actions or states. In this case, we will consider a character with 5 states: IDLE, RUN, JUMP, DIE and FALL. These states can be triggered by keypresses, as described below. The enemy also has its own proprietary follow animation. Player and enemy FSM can be seen in Figures 35 and 36 respectively.

IDLE: This is the default state of the character, in which it is not moving or performing any actions. It can be triggered by releasing all keys or by pressing a key that is not recognized as a command.

RUN: This state represents the character running or moving. It can be triggered by pressing the left or right arrow keys or A and D. The sprite is flipped on the y-axis based on the direction that the player is facing, while still utilising the same animation.

JUMP: This state represents the character jumping. It can be triggered by pressing the up-arrow key, and the animation is then prompted to play via the FSM when the velocity on the y-axis is positive.

DRAINED/DIE: This state represents the character dying or being defeated. It can be triggered by the player either hitting a hazardous trap or contacting the enemies. Figure 34 illustrates the player animation sprites

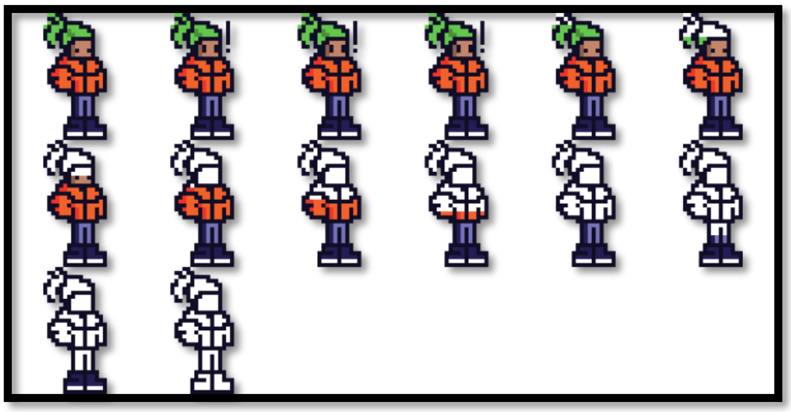
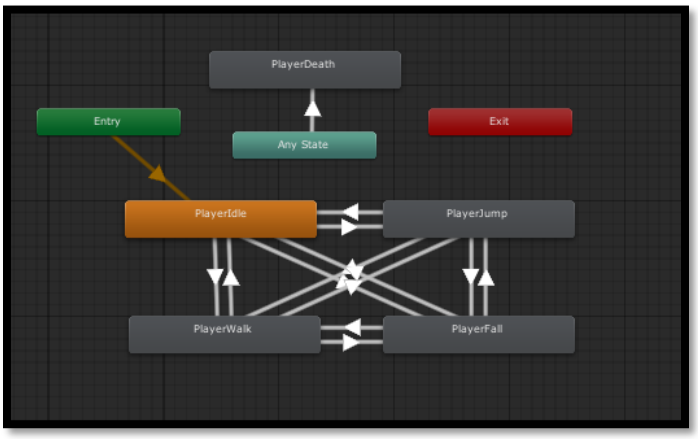
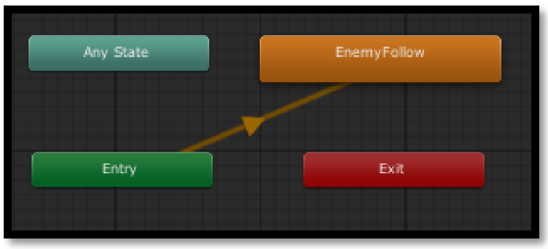


Fig. 34: The Sprite sheet for the player death/drained animation.

FALL: This state represents the character falling downwards, and the animation is then triggered to play via the FSM when the velocity on the y-axis is negative.





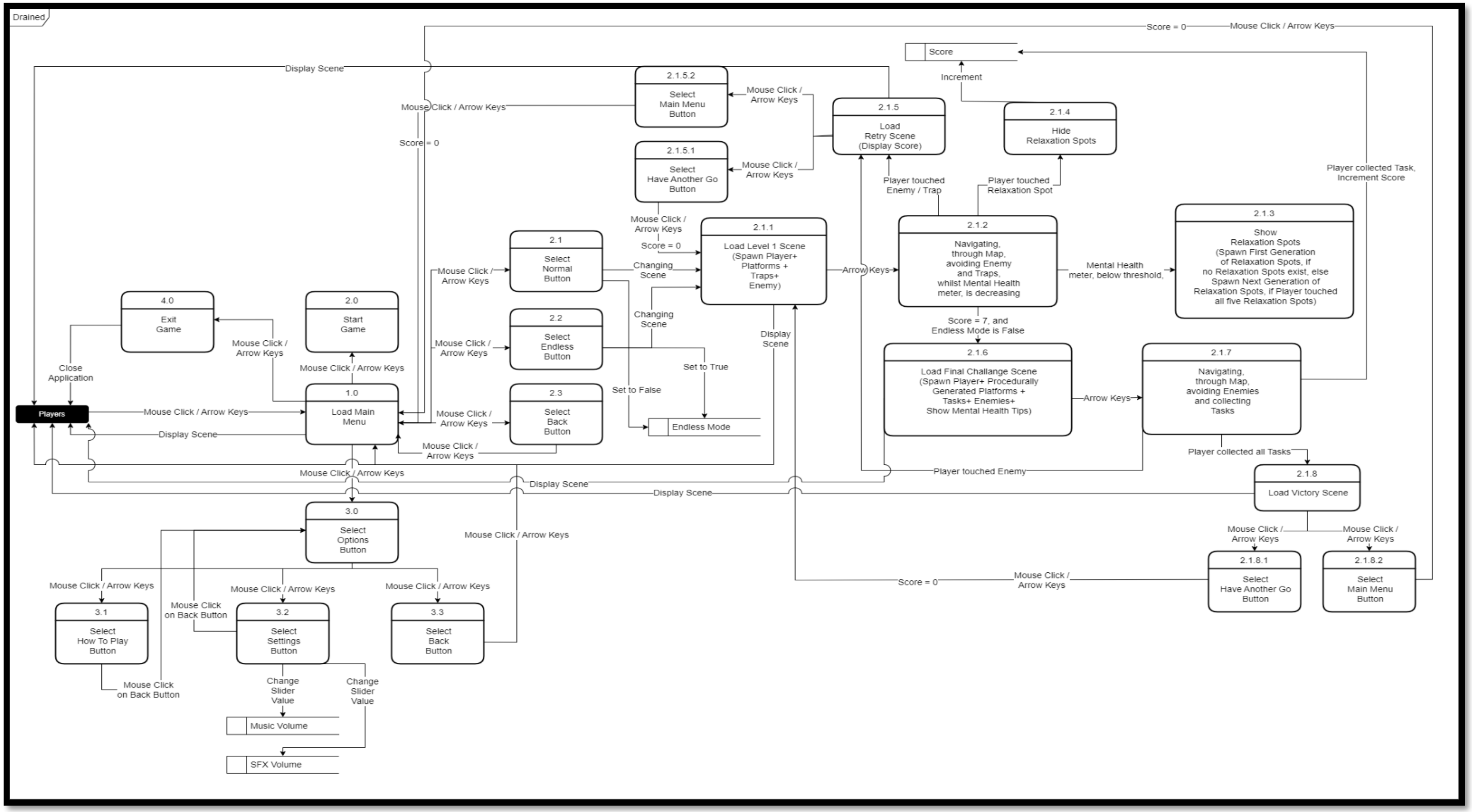
Figs. 35 and 36: Figures showing, Finite State machines for the player and enemies, throughout the game.

FSMs were implemented using nested if-else statements and a state transition table could have been implemented in cohesion with said algorithms, as mentioned in [22].

Level 1 Data Flow Diagram (DFD)

The Data Flow Diagram presented in the next page of this documentation highlights the functionality within *Drained*. It incorporates both the game flow, as well as the manipulation of game variables, such as volume sliders. Conclusively, this DFD shows how the player not only interacts with the environment, but also with the enemies.

Fig. 37: Level 1 Data Flow Diagram (DFD)

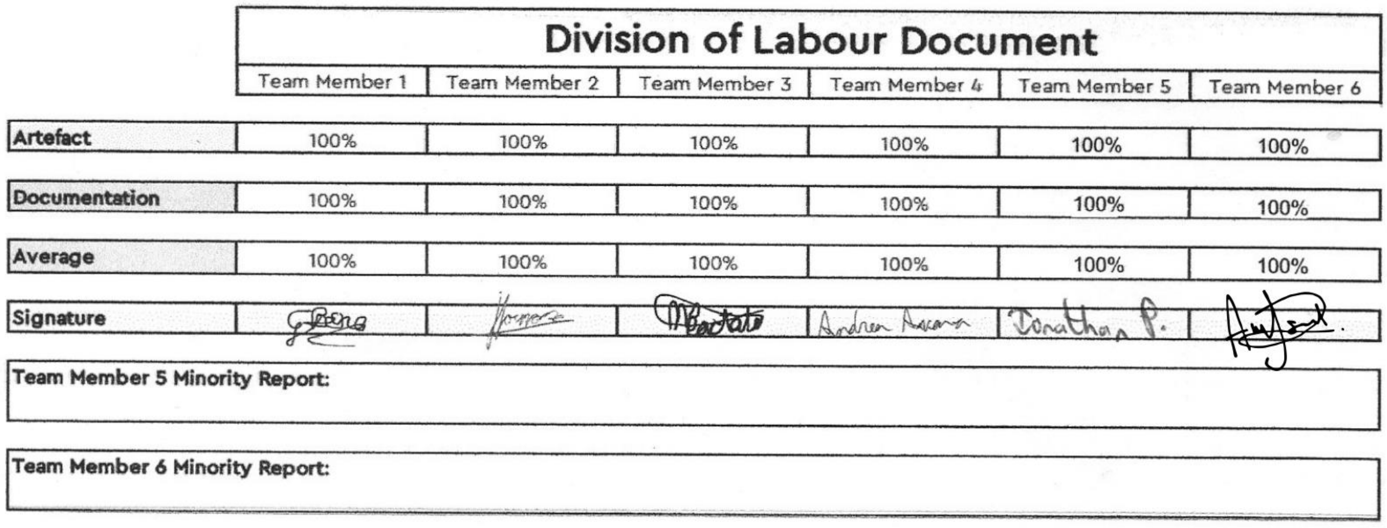

Drained
Players
4.0 Exit Game
2.0 Start Game
1.0 Load Main Menu
3.0 Select Options Button
3.1 Select How To Play Button
3.2 Select Settings Button
3.3 Select Back Button
2.1 Select Normal Button
2.2 Select Endless Button
2.3 Select Back Button
2.1.1 Load Level 1 Scene (Spawn Player+ Platforms + Traps+ Enemy)
2.1.2 Navigating, through Map, avoiding Enemy and Traps, whilst Mental Health meter, is decreasing
2.1.3 Show Relaxation Spots (Spawn First Generation of Relaxation Spots, if no Relaxation Spots exist, else Spawn Next Generation of Relaxation Spots, if Player touched all five Relaxation Spots)
2.1.4 Hide Relaxation Spots
2.1.5 Load Retry Scene (Display Score)
2.1.5.1 Select Have Another Go Button
2.1.5.2 Select Main Menu Button
2.1.6 Load Final Challange Scene (Spawn Player+ Procedurally Generated Platforms + Tasks+ Enemies+ Show Mental Health Tips)
2.1.7 Navigating, through Map, avoiding Enemies and collecting Tasks
2.1.8 Load Victory Scene
2.1.8.1 Select Have Another Go Button
2.1.8.2 Select Main Menu Button
Score
Endless Mode
Music Volume
SFX Volume
Mouse Click / Arrow Keys
Display Scene
Close Application
Score = 0
Changing Scene
Set to True
Set to False
Arrow Keys
Increment
Player touched Enemy / Trap
Player touched Relaxation Spot
Mental Health meter, below threshold,
Player collected Task, Increment Score
Score = 7, and Endless Mode is False
Player collected all Tasks
Player touched Enemy
Mouse Click on Back Button
Change Slider Value

Video

The visualization portion of this assignment can be accessed via either the video file submitted or on YouTube by accessing the following link:

https://www.youtube.com/watch?v=EiGw8lY5pUM

Distribution of Work



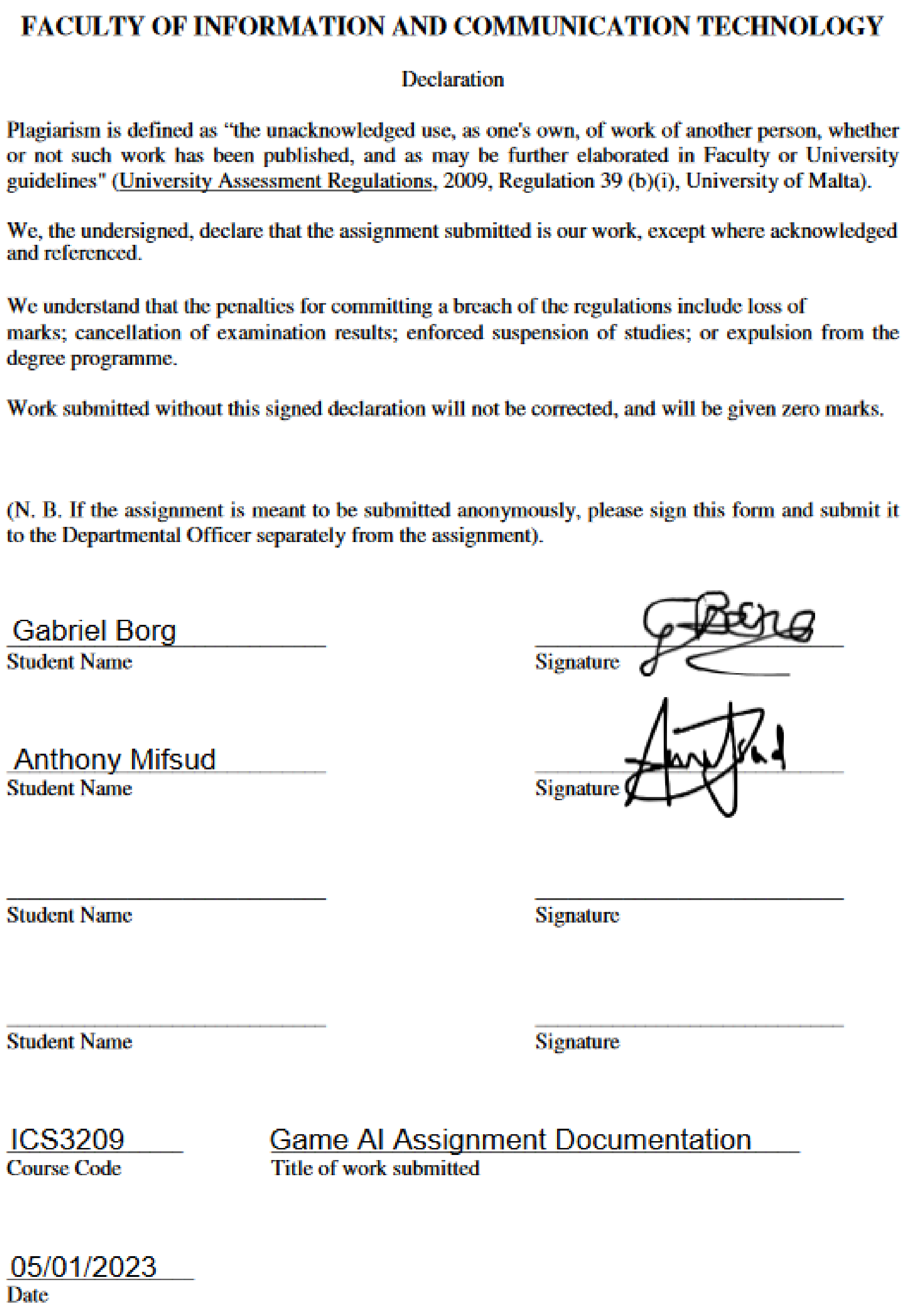
Division of Labour Document

	Team Member 1	Team Member 2	Team Member 3	Team Member 4	Team Member 5	Team Member 6
Artefact	100%	100%	100%	100%	100%	100%
Documentation	100%	100%	100%	100%	100%	100%
Average	100%	100%	100%	100%	100%	100%
Signature	[illegible]	[illegible]	[illegible]	[illegible]	Jonathan P.	[illegible]

Team Member 5 Minority Report:

Team Member 6 Minority Report:

Plagiarism Declaration Forms



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

We, the undersigned, declare that the assignment submitted is our work, except where acknowledged and referenced.

We understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

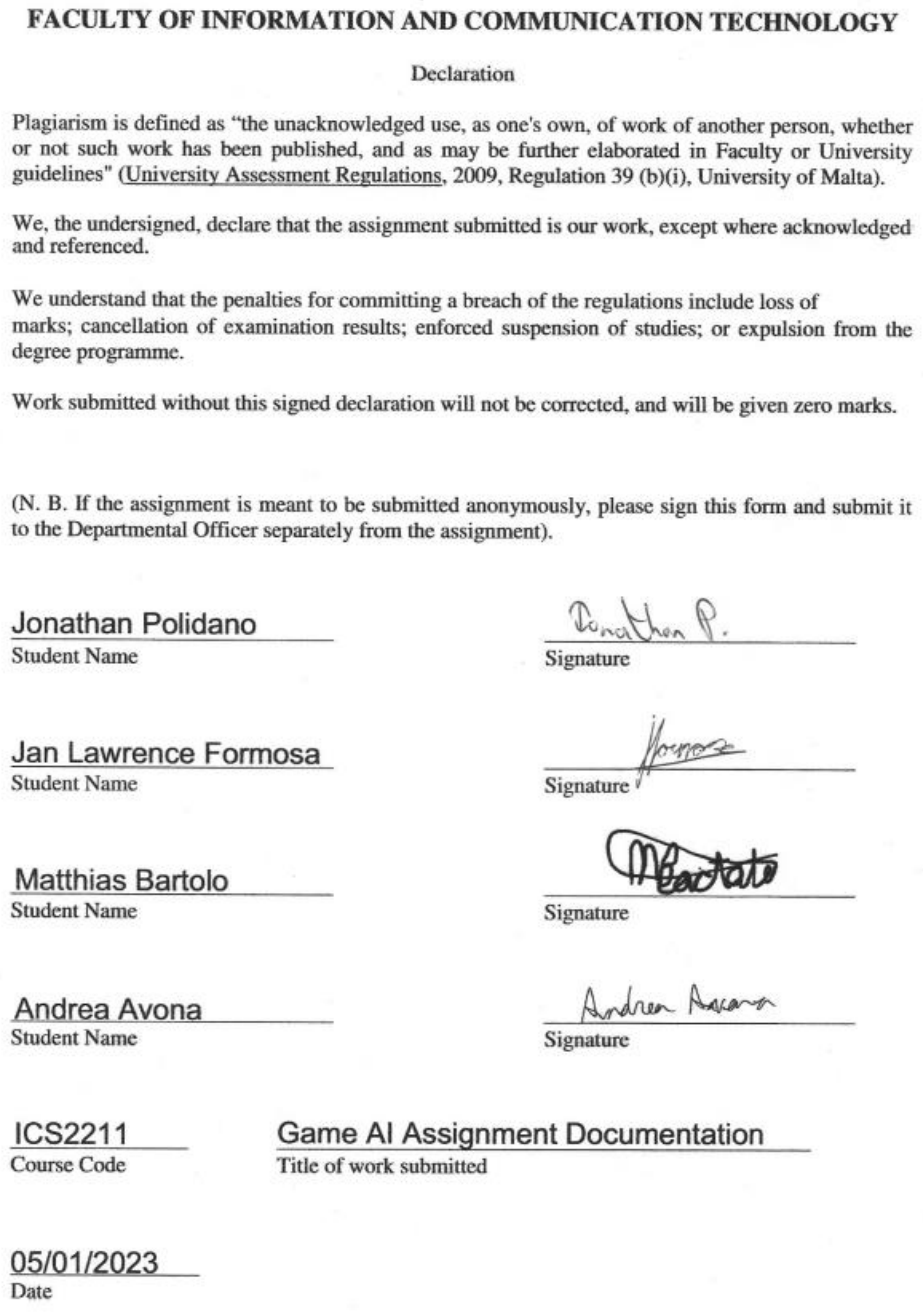
Work submitted without this signed declaration will not be corrected, and will be given zero marks.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Gabriel Borg	[illegible]
Student Name	Signature
Anthony Mifsud	[illegible]
Student Name	Signature
Student Name	Signature
Student Name	Signature

ICS3209	Game AI Assignment Documentation
Course Code	Title of work submitted

05/01/2023
Date



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

We, the undersigned, declare that the assignment submitted is our work, except where acknowledged and referenced.

We understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Jonathan Polidano	Jonathan P.
Student Name	Signature
Jan Lawrence Formosa	[illegible]
Student Name	Signature
Matthias Bartolo	[illegible]
Student Name	Signature
Andrea Avona	[illegible]
Student Name	Signature

ICS2211	Game AI Assignment Documentation
Course Code	Title of work submitted

05/01/2023
Date

References

[1] J. Nguyen, “What Exactly Is Mental Exhaustion? Symptoms, Causes & How To Manage” 2022 [Online]. Available: https://www.mindbodygreen.com/articles/mental-exhaustion [Accessed: 28- Dec- 2022]

[2] S.Waters, “Is your brain tired? Here are 6 ways to treat mental fatigue” 2021[Online]. Available: https://www.betterup.com/blog/mental-fatigue [Accessed: 28- Dec- 2022]

[3] Coding in Flow, Build a 2D Platformer Game in Unity | Unity Beginner Tutorial 2021 [Online video]. Available: https://www.youtube.com/watch?v=Ii-scMenaOQ . [Accessed: 20- Dec- 2022]

[4] Unity Documentation, “Rigidbody” in Version: 2021.3 [Online]. Available: https://docs.unity3d.com/ScriptReference/Rigidbody.html . [Accessed: 20- Dec- 2022]

[5] Unity Documentation, “Input Manager” in Version: 2021.3 [Online]. Available: https://docs.unity3d.com/Manual/class-InputManager.html . [Accessed: 20- Dec- 2022]

[6] Unity Documentation, “Collision2D” in Version: 2021.3 [Online]. Available: https://docs.unity3d.com/ScriptReference/Collision2D.html . [Accessed: 20- Dec- 2022]

[7] Unity Documentation, “Sprites” in Version: 2021.3 [Online]. Available: https://docs.unity3d.com/Manual/Sprites.html . [Accessed: 20- Dec- 2022]

[8] SEGA, “SONIC ORIGINS” 2022 [Online]. Available: https://asia.sega.com/SonicOrigins/en/ [Accessed: 28- Dec- 2022]

[9] Brand Name Audio, 8-Bit Retro Video Game Sound Effects 2 2016 [Online video]. Available: https://youtu.be/7UZQ7NvLNgA?list=PLBtCJCy4TzZTkjWuiIm7RidconrtD_ylj [Accessed: 28- Dec- 2022]

[10] BandLab, “The Future of Music.Here Today.” 2022 [Online]. Available:https://www.bandlab.com . [Accessed: 28- Dec- 2022]

[11] DrPetter, “sfxr” 2012 [Online]. http://drpetter.se/project_sfxr.html . [Accessed: 28- Dec- 2022]

[12] Speed Tutor, Learn UNITY AUDIO 🎧 (The Ultimate Beginner GUIDE) 2021 [Online video]. Available: https://www.youtube.com/watch?v=YnIiMCnAf9E&t=790s [Accessed: 28- Dec- 2022]

[13] Unity Documentation, “PlayerPrefs” in Version: 2021.3 [Online]. Available: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html [Accessed: 28- Dec- 2022]

[14] Greg Eads, “How to use UI Slider to change the volume of Audio Sources across scenes - Unity Tutorial - 2019” 2019 [Online video]. Available: https://www.youtube.com/watch?v=9ROolmPSC70&t=821s [Accessed: 28- Dec- 2022]

[15] V. Mallawaarachchi “Introduction to Genetic Algorithms — Including Example Code” 2017 [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3. [Accessed: 20- Dec- 2022]

[16] Geeks for Geeks, “Genetic Algorithms” 2022 [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/. [Accessed: 20- Dec- 2022]

[17] A. Attard, ICS2211: Genetic Algorithms in Unity, pp. 9-21. 2022 [Online]. Available: https://www.um.edu.mt/vle/mod/resource/view.php?id=963318. [Accessed: 20- Dec- 2022]

[18] Vandrake, “Procedural Generation in Game Development” 2020 [Online]. Available: https://www.davidepesce.com/2020/02/24/procedural-generation-in-game-development/ [Accessed: 28- Dec- 2022]

[19] Geeks for Geeks, “Difference between BFS and DFS” 2022 [Online]. Available: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/ [Accessed: 28- Dec- 2022]

[20] X. Cui , H. Shi, “A*-based Pathfinding in Modern Computer Games”, 2011 [Online]. Available: https://www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games [Accessed: 28- Dec- 2022]

[21] Brackeys, 2D PATHFINDING - Enemy AI in Unity 2019 [Online video]. Available: https://www.youtube.com/watch?v=jvtFUfJ6CP8&t=649s . [Accessed: 20- Dec- 2022]

[22] D. Jagdale, “Finite State Machine in Game Development”, pp. 384–390, 10 2021 [Online]. Available: https://www.researchgate.net/publication/355518086_Finite_State_Machine_in_Game_Development [Accessed: 28- Dec- 2022]