Game AI Project Documentation

ICS2211: Game AI
ICS3209: Advanced Game AI

Emma Gatt
`emma.gatt.21@um.edu.mt`

Kian Parnis
`kian.parnis.20@um.edu.mt`

Liam Bugeja Douglas
`liam.bugeja-douglas.20@um.edu.mt`

Kelsey Bonnici
`kelsey.bonnici.21@um.edu.mt`

Shaizel Bezzina
`shaizel.bezzina.21@um.edu.mt`

Olesia Shtanko
`olesia.shtanko.21@um.edu.mt`

B.Sc. (Hons) Artificial Intelligence
University of Malta

Contents

The game unity package and executables can be found *here*.
The video can be found *here*.

1 Introduction

As our GameJam 2022 entry, we developed “State of mind”. A game designed using unity game engine and scripted using the language C#.

Our theme for this year’s GameJam was Mental Health. In line with this, we came up with the idea of creating a single player game – an elderly man. The player who has Alzheimer’s disease must work through the different levels, to see the different stages of Alzheimer’s.

The aim of our game is to collect as many memories as possible without the character being hit by enemies, in order see how Alzheimer’s disease affects the memories of someone.

In order for the character to move, the person playing the game must choose his preference from either using the wasd keys, or the arrow keys along with the shift key to dash.

State of mind is set in the mind of the elderly man. The mind is made to look like a maze, in which the player has to navigate through to collect the memories and avoid the enemies.

It is also worth noticing that all of the artwork was created by our designers, whilst all sounds were chosen to ensure a captivating experience.

We believe that despite the short time frame given for developing this game, the result is enjoyable whilst promoting values such as respect and teamwork.

2 Conceptual documentation of the game

2.1 *State of Mind* Addressing the Theme of Mental Health

Alzheimer's disease is a type of dementia that affects the memory, thinking, problem-solving abilities, and behavior. Alzheimer's is a progressive brain disorder, meaning that it worsens over time. The majority of individuals affected by Alzheimer's disease are older people. This disease has 7 stages where each stage shows worse symptoms than the one before. The 7 stages are no impairment, very mild decline, mild decline, moderate decline, moderately severe decline, severe decline, and very severe decline. **Recognition of Alzheimers disease**

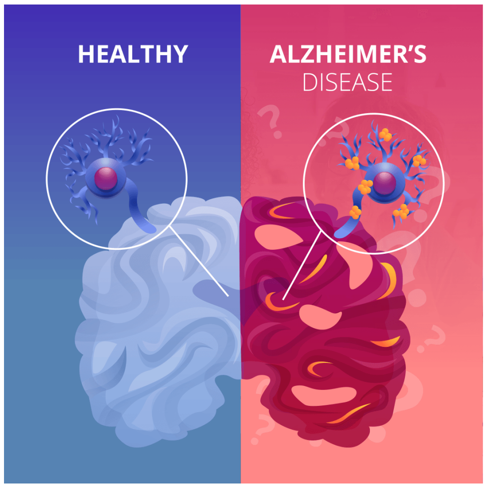


Figure 1

Mental health is the state of a person's total psychological well-being, which includes their ability to operate emotionally, socially, and psychologically. Alzheimer's disease can have a significant impact on a person's mental health. Confusion, frustration, and anxiety are common side effects of cognitive decline, as well as personality and behavioural changes, which can cause negative effects or the individual's caregivers and family due to being an emotional stressful journey.

2.2 The Game Mechanics

Explained by Dave Eng, Eng, 2020 game mechanics are essentially "rules and procedures" that outline the games' reaction to players' decisions and actions. It is generalized by Stephen Trinh that in a quality game there are four main elements of game mechanics: Quantity, State, Spatial and Action, Trinh, 2020. The purpose of said mechanics is to enhance the gaming experience of players by guiding and immersing into the concept of the game if implemented thoughtfully. In *State of Mind* several of mentioned game mechanics were employed to guide players through the ropes of the game.

Figure 2: Main menu

Firstly, quantity, a mechanic that is represented as numbers can be noted upon playing the game. The player has 6 memories to pick up along the maze, as each is found, the number of collected to total increases. Mental health is represented by a bar that depletes on colliding with the enemy. On average it takes 2 to 3 hits depending on the difficulty of the level.

Secondly, spatial mechanics are also implemented. The randomly generated map as well as the position of the player on the map, are a few of the examples. Additionally, it is considered that the position of the memory objects, collision with them and the character itself, make part of this element of game mechanics. The enemies are thoughtfully modelled to increase in number as the game progresses in order to give the player the right level of difficulty on each level of the game.

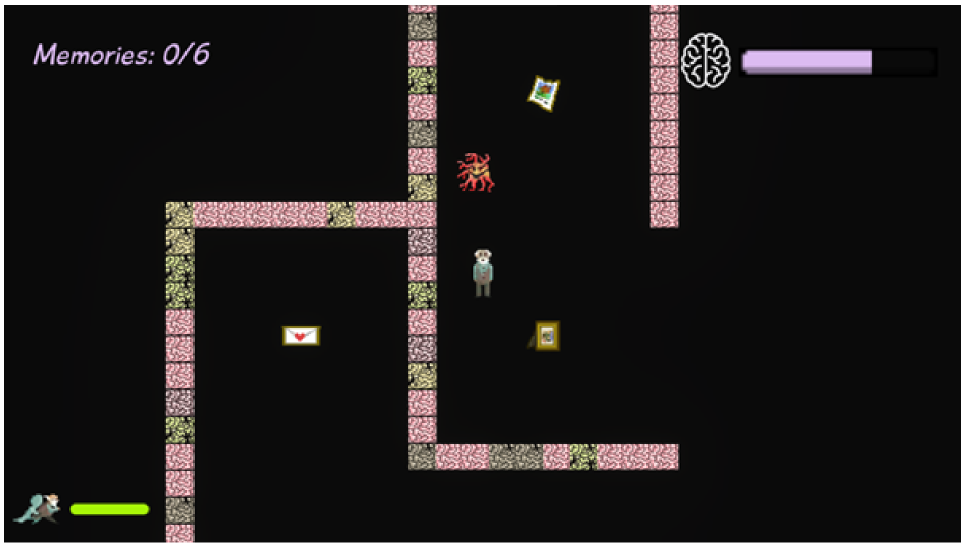


Figure 3: Display of game mechanics

The third element of game mechanics, state, is represented by the collision of the player with the enemies, paired with their respective animations, which will be discusses in further detail further on. The collision with memories is yet another example of state mechanics. Each collectable represents a valuable memory to the player that the character tries to recall in order to engage the player into his story while pausing the main action of the game. On top of that, there are game states. There are four game states implemented, namely, the main menu, the playable levels, the recalling of memories when colliding with the collectables as mentioned above, and the end screen. An interesting mechanic is that on depletion of mental health, the player does not die. Instead, the maze is rotated to confuse the player, further promoting the experience of Alzheimer's disease and how the person living with this progressive disease feels on a daily basis.

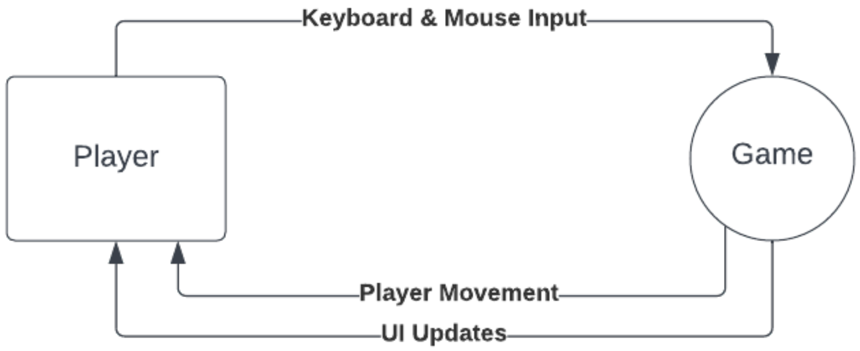
Lastly, the action game mechanics. These mechanics provide change in order to keep the game exciting. Several examples include interact-able collectables, found around the maze; the enemy attacking the player as well as the players' ability to evade the enemy and speed up by using the boost mechanic. A meaningful and important mechanic , as mentioned previously, is that the enemies, plaques, deplete the players mental health, worsening his condition. The game encourages the player to learn and appreciate the struggle of someone with Alzheimer's as well as prompt to be patient and lenient similarly to how the game does by confusing the player instead of ending the play-through.

Game mechanics are the building blocks of game design and are a crucial component of the overall experience, demanding careful thought,Wallace, 2013. Each of the four game mechanics explained help the players, keep them engaged, and immerse them into the first-hand experience the game is based on, mental health.

3 Outline of the various modules and Game Components

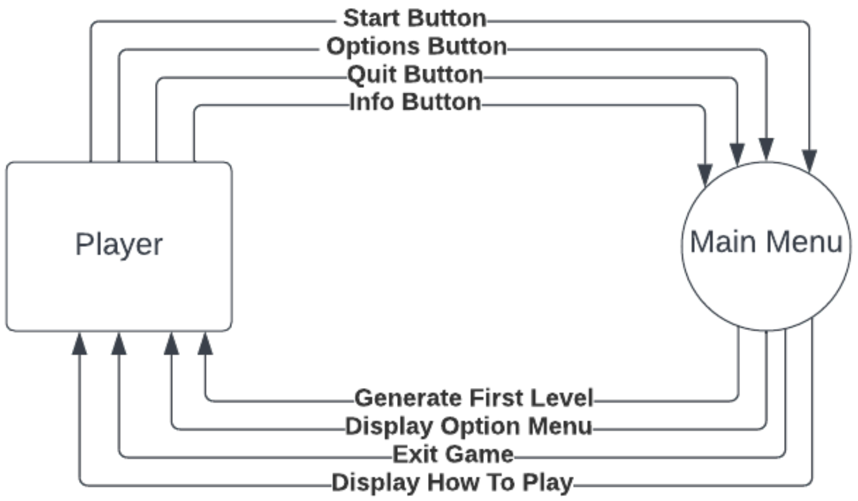
3.1 Basic Gameplay

The game is a single-player experience that relies on keyboard and mouse interaction. The goal of the game is to collect all of the collectibles spread around the stages while avoiding enemies. To complete the game, the player must traverse through stages, avoiding enemies and collecting collectibles all while managing their mental wellbeing.



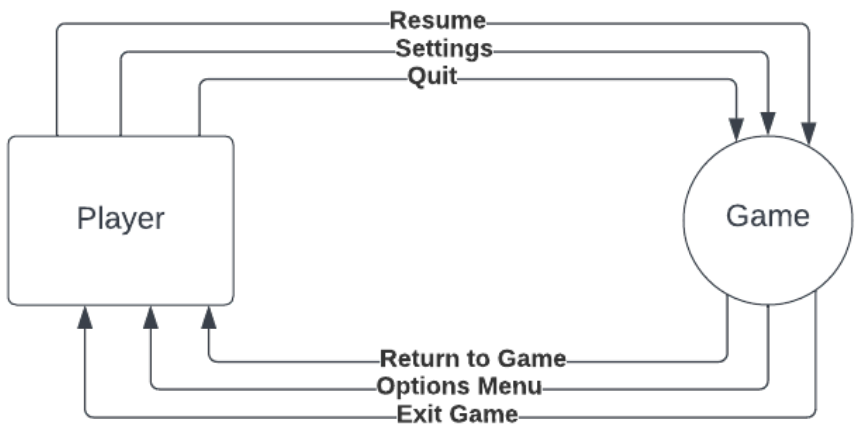
3.2 The Main Menu

The main menu of the game features several options for the player to choose from, including ”New Start,” ”Options,” ”Quit,” and ”Info”. The ”New Start” allows the player to begin the game. The ”Options Menu” allows the player to adjust certain game settings such as volume, graphics and a toggle for full screen is also available. The ”Info” option provides basic information on how to play the game which mainly includes instructions on movement and game objectives. It also includes information about the movement keybinds as well as the dash keybind. Furthermore, it contains information about the collectables and the including a visual representation of them.



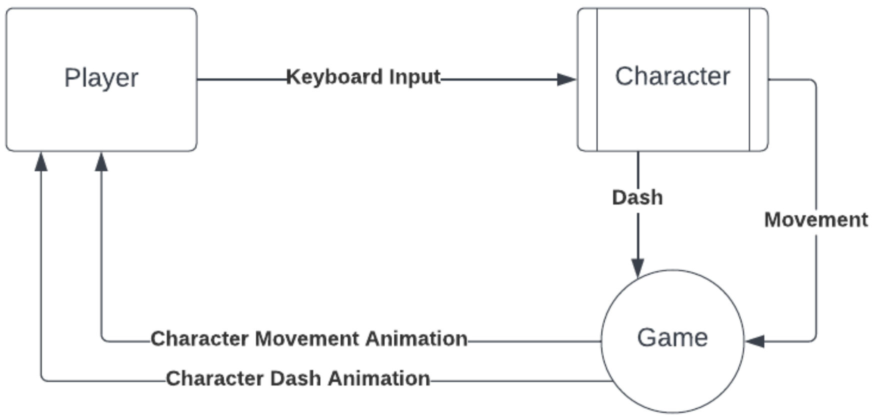
3.3 The Pause Menu

When the player presses the pause button, the game is halted and a pause menu appears on the screen. The pause menu provides the player with three options, which include ”Resume,” ”Setting,” and ”Quit.” When y the ”Resume” button is pressed, the game resumes from where it was halted. The ”Settings” option displays the settings that the player is able t0 modify; also, the settings menu is a replica of the options tab from the main menu. Finally, the user has the option to exit the game by pressing the ”Quit” button.



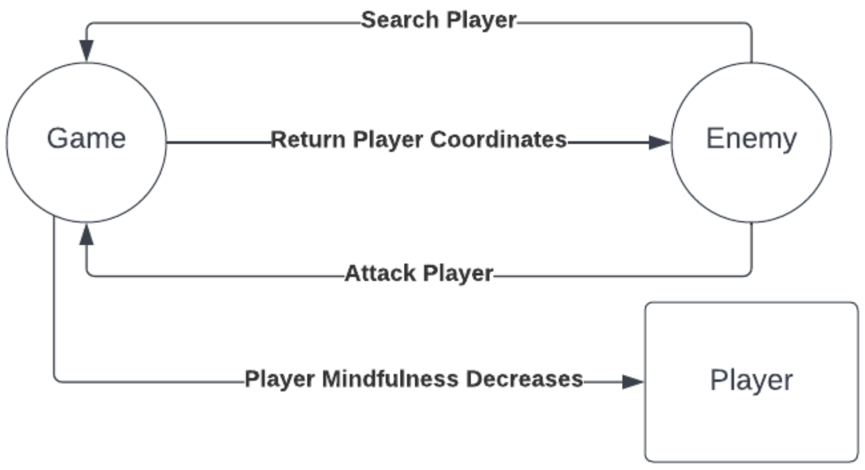
3.4 The Character

The user controls a character in the game which can be controlled using the keyboard. The player can navigate the levels by using the arrow keys or WASD keys to move the character along the x and y axes. The character can also perform a dash move, which is triggered by pressing the shift button. This lets the character to move swiftly in the direction they are facing, which is important for dodging enemies or reach locations at a faster pace. Because the dash move has a brief cooldown period after use, the player must use it wisely.



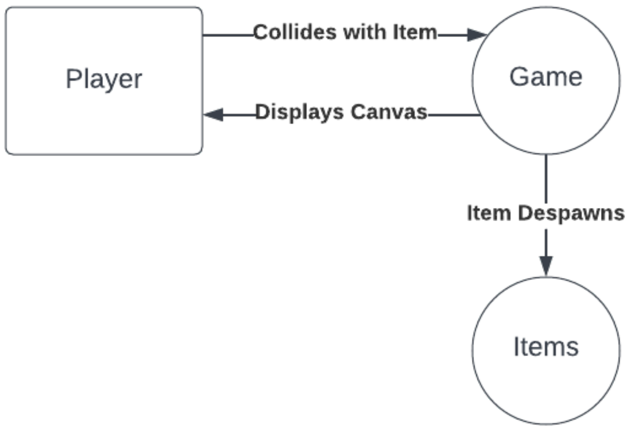
3.5 Enemies

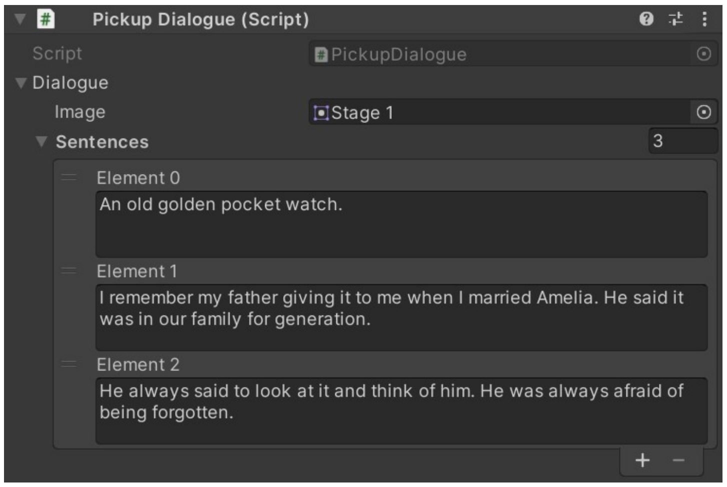
The patrol system allows enemies to cover a set region of the game, whereas the A* algorithm is utilized to follow the player once it reaches a particular distance to the player. The techniques will be further analyzed in the AI feature section.



3.6 Consumables

The player can gather several collectibles placed across the levels of the game. A canvas appears on the screen when the player obtains one of these items. The canvas depicts an image or artwork and a few lines of dialogue relating to the recently acquired object.
The dialogue system was implemented using a DialogueManager for each scene and having the respective dialogue attached to each asset "How to make a Dialogue System in Unity", 2017. When all collectibles have been acquired, the player progresses to the next stage.




Pickup Dialogue (Script)

Script: PickupDialogue

Dialogue

Image: Stage 1

Sentences: 3

Element 0: An old golden pocket watch.

Element 1: I remember my father giving it to me when I married Amelia. He said it was in our family for generation.

Element 2: He always said to look at it and think of him. He was always afraid of being forgotten.

Figure 4: Dialogue attached to an asset

4 Detailed description of AI features

4.1 Maze Generator

As the game was centered around the theme of feeling trapped within one's own mind, slowly progressing through different stages of Alzheimer's disease the main idea to capture this sense of entrapment was to fundamentally build the game utilizing a maze generator. With this, the player would get a sense of feeling lost, trying to remember where they came from and where they are going, which captures the same types of feeling a person with Alzheimer's would feel Wenk et al., 2003; Scheltens et al., 2016; Goedert and Spillantini, 2006.

With this, research was carried out to find the different means one can implement a maze generator. Numerous Maze Generation algorithms exist Kozlova et al., 2015; Foltin, 2011, such as Randomized Kruskal's algorithm, Randomized Prim's Algorithm, and Recursive Backtracking. Recursive backtracking was ultimately chosen to be implemented due to the time-bound of the event. Since GameJam is an event that lasts for only 48 Hours, the lower complexity that Recursive backtracking takes to be implemented and debugged fits within a reasonable time-bound. With that said, since the size of the maze was kept at smaller sizes no noticeable performance tradeoffs were encountered which may have lowered the overall game-play feel.

Recursive backtracking is based on the Depth First Search (DFS) technique.

The DFS algorithm rambles around the graph in a depth-focused manner. It denotes that, whenever possible, the following visited vertex is a child of the previously visited vertex. Otherwise, if the most recently visited vertex doesn't have any more accessible children, the algorithm goes back to the previously visited vertex and tries again. Once the algorithm has visited each of the vertices, it is completed. A spanning tree is finally produced in the event that the method develops a tree while running (by recording and storing visited graph vertices and graph edges).
The following is the Pseudo-code that outlines the Depth First Search (DFS) Algorithm:

Algorithm 1: Depth First Search

```
Data: G: The graph stored in an adjacency list, Root: The starting node
Result: Prints all nodes inside the graph in DFS order
$visited \leftarrow false$;
$stack \leftarrow \{\}$;
while $\neg stack.empty()$ do
    $u \leftarrow stack.top()$;
    $stack.pop()$;
    if $visited|u| = true$ then
        continue;
    end
    $print(u)$;
    for $v \in G[u]$ do
        if $visited|u| = false$ then
            $DFS(u)$;
        end
    end
end
```

The operation of the recursive backtracking algorithm is similar. It follows the starting point across the entire graph, occasionally going backwards. The crucial component is randomization. The first randomization step takes place when a random cell is selected as the algorithm's starting point. When selecting a child vertex from the current cell to slice into, another randomization occurs.

The following is the pseudocode of how this algorithm was implemented in the game.

With this implementation, a random maze would be generated with a size specified to the size of x and y. The maze would be generated every time a player goes to a new stage by picking up all pickups needed to progress. To give a better feel-

Algorithm 2: Recursive Backtracking Implementation

```
Data: Initial grid specified to the maze's specified grid size $x, y$
Result: Generates a Maze of size $x, y$
$currentPath \leftarrow randomNode$ ;  /* A node from grid size x, y */
while $completedNodes < nodesCount$ do
    $direction \leftarrow getNextPossibleDirection()$;
    if $direction.Count > 0$ then
        $removeWall \leftarrow direction$;
        $currentPath \leftarrow chosenNode$;
    else
        $completedNode \leftarrow currentPath.top()$;
        $currentPath.pop()$;
    end
end
```

ing of the disheartening progression of Alzheimer's, at every new stage, the size of the maze would be increased slightly, before being generated. Additionally, each individual "node" wall where given an opacity of 0. This would gradually increase to 1 once the player enters the node. This was carried out to give the feeling that the playable character is trying to remember what paths they took, so he could reach his core memories (pickups) faster.

4.1.1 Examples of generation

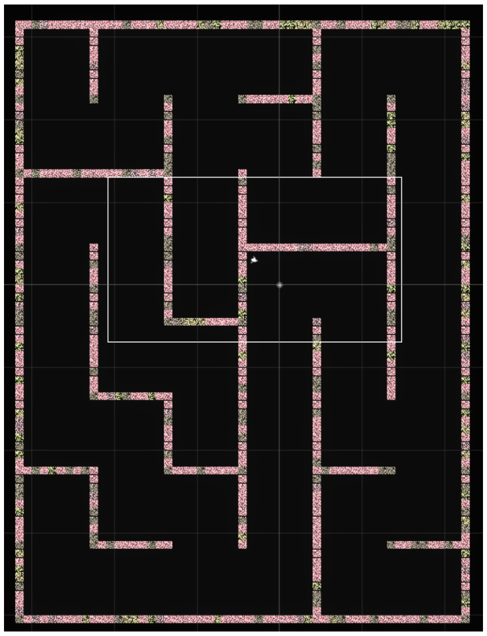


Figure 5: Generation Example 1

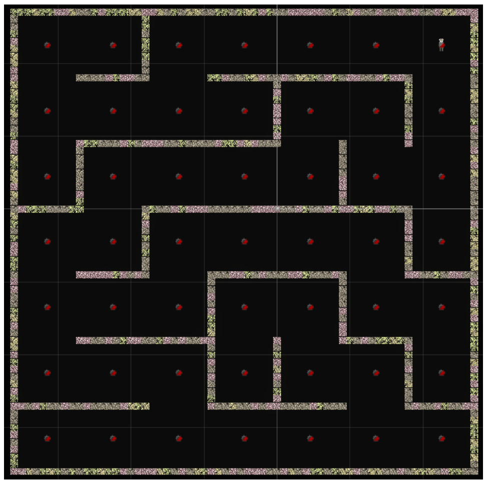


Figure 6: Generation Example 2

4.2 A* Search Algorithm

In order for the adversaries to easily navigate toward the player while being aware of the walls in their way, a pathfinding algorithm was needed. This algorithm needs to be quick and accurate in order for the enemies to compute the best route to the player without becoming entangled by obstacles or taking an excessively long route that would make it easy for the player to evade them. The A* search algorithm was chosen as a result because of how quickly and effectively it can discover a path.

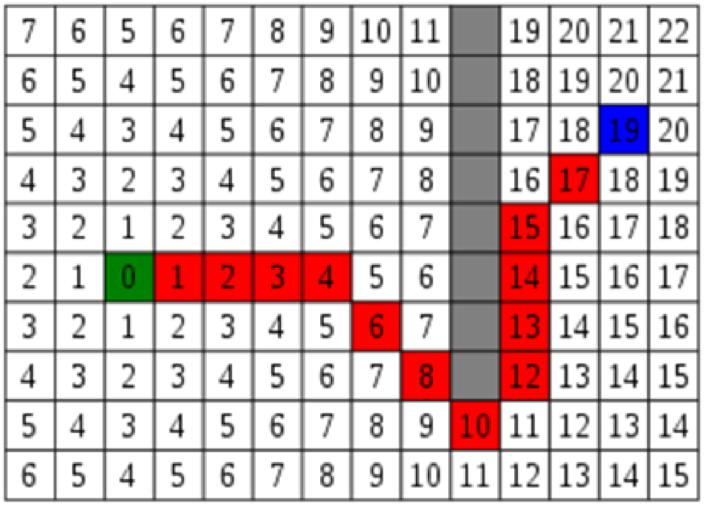


Figure 7: Illustration of A* Search Swift, 2020a

Due to how easily it can be integrated into other applications, A* is one of the most well-known pathfinding algorithms out there Swift, 2020b; Gross et al., 2018. A* uses a heuristic to steer itself and looks for the quickest route between two points. The efficiency in pathfinding is due to a combination of shortest paths complimented by the use of heuristics. Two algorithms to note lack behind due to the fact that they employ only one of these two methods, such as Dijkstra's Algorithm in the case of finding the shortest path and Greedy Best-First-Search in the case of the heuristic function Swift, 2020b.

4.2.1 The heuristic function

The predicted path with the lowest cost between two points is determined using a heuristic function. Depending on how the cost is determined in the specific application where A* is being used, a variety of heuristic functions can be selected.

Heuristics like Euclidean distance, Manhattan distance, and Great Circle distance can be utilized when the cost to move between nodes is directly proportional to the distance between them. n.d., -b.

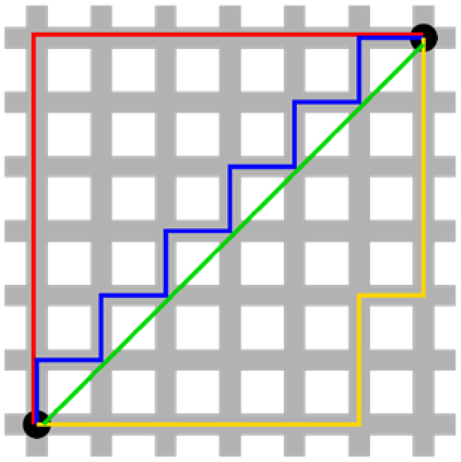


Figure 8: Manhattan Distance Heuristic “A* search”, n.d.

4.2.2 A* algorithm details

The search space is contained in a two-dimensional plane where the algorithm operates on a grid. The cost of traveling between the grid’s nodes, which is divided into a number of them, is used as a yardstick to decide which way is the shortest. A* must eliminate any inaccessible nodes from the path it finds in order to make it possible to travel along it. When determining how distant the path to the objective will be from a given node, the method first takes into account all of the nodes that are close to the beginning node and uses the heuristic function to estimate that distance. The algorithm then advances the point towards that path and iteratively repeats the first two steps until it reaches the desired outcome.“A* search”, n.d.

A* utilizes two values: h(n), which is a heuristic that estimates the cost from the vertex ’n’ to the goal, and g(n), which represents the precise cost of any path from the starting point to a vertex ’n’. A* applies the equation f(n) = g(n) + h(n) to calculate which node offers the shortest path to the destination by combining these two components. The following is the Pseudo-code that outlines the A* Algorithm:

Algorithm 3: The A* Algorithm

```
Data: graph
Data: startNode
Data: targetNode
Result: Path
for $node - in - graph$ do
    $node.score \leftarrow \infty$;;
    $node.heuristicScore \leftarrow \infty$;
    $node.visited \leftarrow \infty$;
end
$startNode.score \leftarrow 0$;
$startNode.heuristicScore \leftarrow 0$;
while $true$ do
    $currentNode \leftarrow lowestNodeScore(graph)$;
    $currentNode.visited \leftarrow true$;
    for $nextNode - in - currentnode.neighbors$ do
        if $nextNode.visited == false$ then
            $newScore \leftarrow calculateScore(currentNode, nextNode)$;
            if $newScore < nextNode.score$ then
                $nextNode.score \leftarrow newScore$;
                $nextNode.heuristicScore \leftarrow newScore +$
                  $calculateHeuristicScore(nextNode, targetNode)$;
                $nextNode.routeToNode \leftarrow currentNode$;
            end
        end
    end
    if $currentNode == targetNode$ then
        return$(buildPath(targetNode))$;
    end
    if $lowestNodeScore(graph).score == \infty$ then
        throw$(noPathExists)$;
    end
end
```

Algorithm 4: lowestNodeScore

```
Data: graph
Result: node
$result \leftarrow null$;
for $nodeingraph$ do
    if $node.visited == False - AND - node.heuristicScore <$
      $result.heuristicScore$ then
        $result \leftarrow node$;
    end
    return($result$);
end
```

4.2.3 Implementation within the game in Unity

Within the game itself, an implementation of A*[1] was used. As a result, it was possible to download the complete package of tools and materials from their website and modify them before using them in the game. These scripts were then slightly modified to extend the behavior needing to be exhibited by the adversaries. This added behavior was the possibility for the AI to lose interest / not path-find towards the player and wander around the maze instead if they are a certain distance away from the player. This was achieved by giving every maze ”node” a point that can be selected by the A* Algorithm. If the player is away from the adversaries AI would pick a random point to pathfind to, if they arrive at a very short distance to the point, then they would pick a new point and start the pathfinding process anew. If at any point the player is within distance, the adversary would then start pathfinding towards the player instead.

[1] https://arongranberg.com/astar/

5 Animations

Animation in 2D games is vital because it adds realism, immersion, and emotion to the game. It brings characters and objects to life by animating them in a believable and natural manner. Animation was used in this game to send information to the user, such as when the character moves or is hit by an enemy. Custom sprite sheets were created by for the motion and hit registration of all the game characters. Figures 7 and 8 contain the sprite sheets of the main character and enemies respectively.

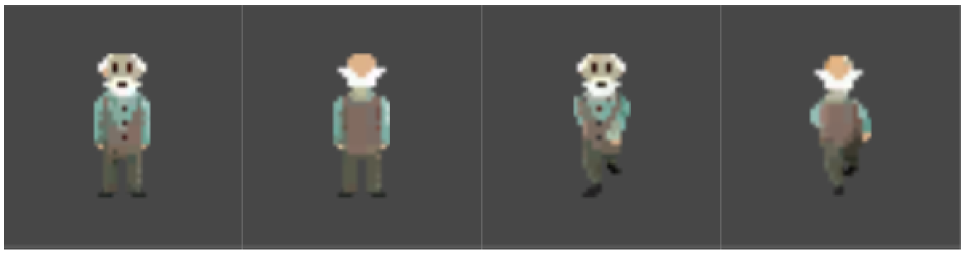


Figure 9: Player Sprite Sheet

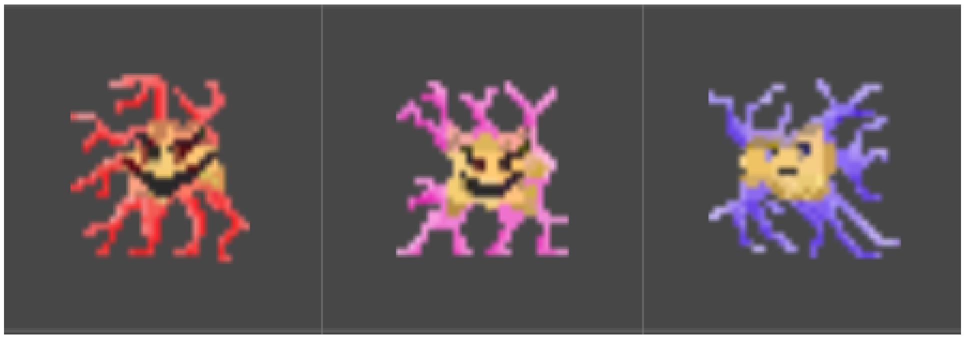


Figure 10: Enemy Sprite Sheet

5.1 Player Animations

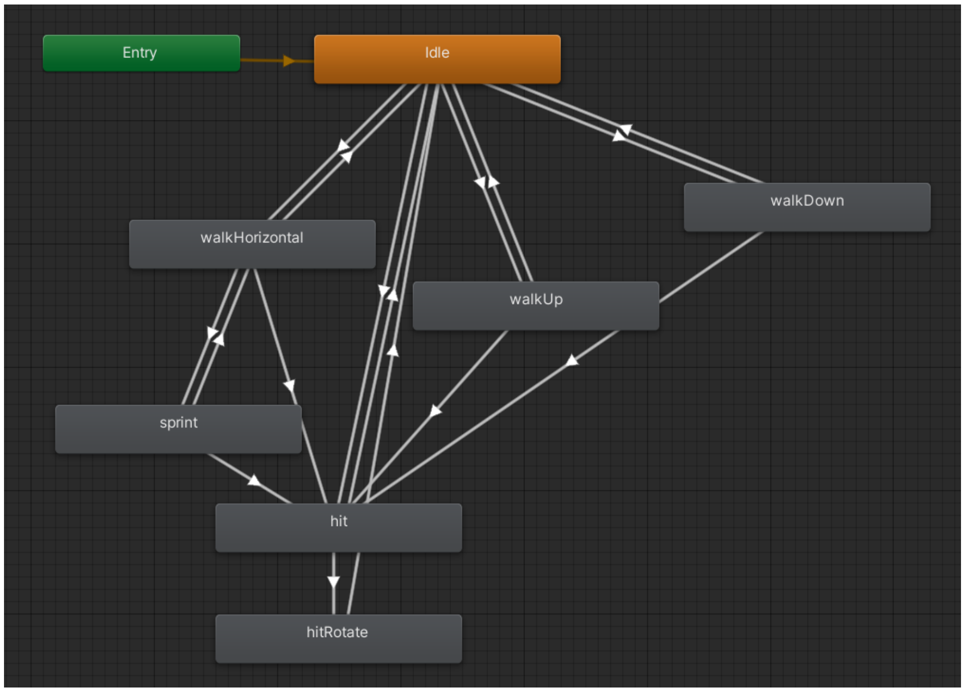


Figure 11: Main Character Animation State

The player state machine begins in a ”Idle” state and is changed by the player’s movement or an enemy attacking the player.When the player character moves left or right, the ”Walk Horizontal” state is activated. This stage is triggered by the player inputting horizontal movement, and it changes the main character’s sprite to the corresponding sprite animation.When the player moves up or down, the ”Walk Down” and ”Walk Up” states are utilized. These states are activated when the player enters vertical movement, and they modify the character sprite animation to the relevant one.The player enters the ”Sprint” state by pressing the sprint button. This state gives the player a white trail animation and temporarily increases their speed. The white trial animation is used to visually represent the sudden change in player speed.When the player character is attacked by an enemy, the ”Hit” state is activated. This stage modifies the character’s animation and plays a sound effect in response to the collision. Furthermore, the player’s mental well-being decreases in this state. Finally the ”Hit Rotate” is utilized when the player’s mental well-being is reduced to zero. This state rotates the map and is intended to confuse the player.

5.2 Enemy Animations

In the game, enemies have two main types of movement animations: patrolling and following the player. The patrolling animation is employed when the opponent is going along a predetermined course and is not actively pursuing the player. While the following animation is utilised when the opponent is actively chasing the player. Furthermore, the different movement states also have different sprite visuals. This alerts the player when an enemy is aware of their presence, allowing them to react appropriately.

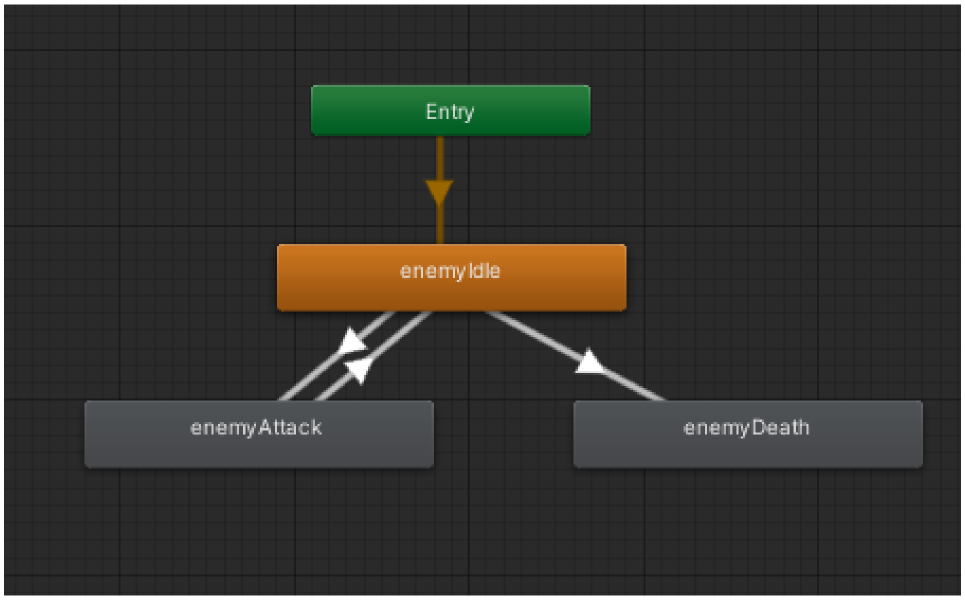


Figure 12: Enemies in the game

The ”enemyIdle” state is used when the enemy is patrolling the map. Whilst the ”enemyAttack” is deployed when the enemy detects and follows the player. Finally, the ”enemyDeath” is used when the enemy collides with the player and despawns.

References

Eng, D. (2020). Game mechanics. https://medium.com/@davengdesign/game-mechanics-3f2b338047aa

Trinh, S. (2020). The 4 elements of game mechanics. https://www.gamedeveloper.com/disciplines/the-4-elements-of-game-mechanics

Wallace, A. (2013). Game design: The importance of game mechanics. https://www.gameskinny.com/rueda/game-design-the-importance-of-game-mechanics

How to make a dialogue system in unity. (2017). https://www.youtube.com/watch?v=_nRzoTzeyxU

Wenk, G. L., et al. (2003). Neuropathologic changes in alzheimer's disease. *Journal of Clinical Psychiatry*, *64*, 7–10.

Scheltens, P., Blennow, K., Breteler, M. M. B., de Strooper, B., Frisoni, G. B., Salloway, S., & Van der Flier, W. M. (2016). Alzheimer's disease. *Lancet (London, England)*, *388*(10043), 505—517. https://doi.org/10.1016/s0140-6736(15)01124-1

Goedert, M., & Spillantini, M. G. (2006). A century of alzheimer's disease. *science*, *314*(5800), 777–781.

Kozlova, A., Brown, J. A., & Reading, E. Examination of representational expression in maze generation algorithms. In: In *2015 ieee conference on computational intelligence and games (cig)*. IEEE. 2015, 532–533.

Foltin, M. (2011). Automated maze generation and human interaction. *Brno: Masaryk University Faculty Of Informatics*.

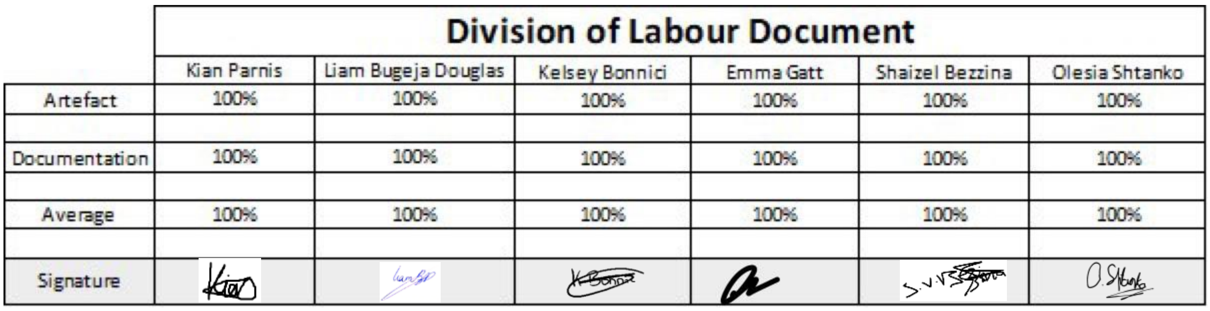
Swift, N. (2020a). Easy a* (star) pathfinding. https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

Swift, N. (2020b). Easy a* (star) pathfinding. https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

Gross, J. L., Yellen, J., & Anderson, M. (2018). *Graph theory and its applications*. Chapman; Hall/CRC.

(n.d. -b). https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&amp;stage=all

A* search. (n.d.). https://brilliant.org/wiki/a-star-search/



Division of Labour Document

	Kian Parnis	Liam Bugeja Douglas	Kelsey Bonnici	Emma Gatt	Shaizel Bezzina	Olesia Shtanko
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]	[illegible]	[illegible]

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Kian Parnis	[illegible]
Student Name	Signature
Liam Bugeja Douglas	[illegible]
Student Name	Signature
Kelsey Bonnici	[illegible]
Student Name	Signature
Emma Gatt	[illegible]
Student Name	Signature
Shaizel Bezzina	[illegible]
Student Name	Signature
Olesia Shtanko	[illegible]
Student Name	Signature

ICS2211/ICS3209	Game AI Project Documentation
Course Code	Title of work submitted
20/01/2023	
Date	