# 

# Title: Recreate a Classic Arcade Game: PacMan

# Student: Fahad Hussain

# Degree: BSc Computer Science

# Module Code: CE301

# Supervisor: Dr Morteza Varasteh

# Final Word Count: 8668

**Table of contents**

Acknowledgements ………………………………….………………………4

Abstract ………………………………….………………………………….4

Key Terms ………………………………….………………………………5-6

**Literature Survey**

1.0 Introduction …………………………………………………….………7-8

1.1 The significance of Pac-Man …………………………..……….8-9

1.2 Gameplay ……………………………………………………… 9

1.3 Why are these games still playable? ………………….……….. 9-10

2.0 - Artificial Intelligence ………………………………………..………. 10 -11

2.1 What is artificial intelligence? ………………………………….11

**Technical Documentation**

3.0 - Game Design ………………………………….…………….…………..11

3.1 - Sprite Design ………………………………….…………………11-12

3.2 - Designing Main Menu ………………………………………….12

3.3 - Language of Choice …………………………………………….12-13

3.3.1 - So, Why Unity? ……………………………………….13

4.0 - Board Class ………………………………….………………….………13-14

4.1 - Setting up the board parent class ……………………….………14-15

4.2 - Creating Objects within the Grid ………………..……….……..15

4.3 – Vectors ………………………………….………………………15-16

4.2 - Point movement ………………………………….………….….16-17

5.0 - Pacman Class………………………………….…………………………17

5.1 - Setting the parent class ………………………………………….17

5.2 - Moving PacMan around the maze ………………………………18

5.3 - Animating PacMan …………………………………………….18-.19

5.4 - Eating pellets …………………………………………………….19

5.5 - Containing PacMan within the Maze ……………………………19

6.0 - Ghost Class ………………………………….………………………… 20

6.1 - Introducing the Ghosts …………………………………………20

6.2 - Setting up the Parent Ghost Class……………………………….21-22  
 6.3 - The Ghost House………………………………………………..22-23  
 6.4 - Red Ghost’s Artificial Intelligence Class……………………….24  
 6.5 - Blue Ghost’s Artificial Intelligence Class……………………..25  
 6.6 - Orange Ghost’s Artificial Intelligence Class……………………25  
 6.7 - Pink Ghost’s Artificial Intelligence Class……………………..25

6.8 - Ghost Animation Class……………………………………….25-26

6.9 - Finite State Machines…………………………………………27

6.9.1 - Ways of implementing FSM into the project: ….…..28  
 6.9.2 -How Finite State Machines are programmed ………28 -30

6.10- The Ghost States Class …………………………………..30-31  
 6.11 - The Ghost Direction Decision Class ……………………21-32  
 6.12 - How Collision Works in this Game. ……………………32-33

Management

7.0 - The Board /Manager Game Object ………………………………33-34

7.1 - The Purpose of Board Class/”Manager” …………………34

8.0 - Testing ………………………………………………………….35  
9.0 - Project Management ……………………………………………35

9.1 – Methodology …………………………………………..35

10. 0– Conclusion ……………………………………………………36

11. 0 Bibliography …………………………………………………….37-42

12. 0 Appendix ……………………………………………………….43-47

Acknowledgements

I would like to thank my department and the modules and opportunities that have been made available. To Albert Solomon library and all their staff, thank you for being open almost every day where I have spent my last few months.

Abstract

The aim of this project was to recreate a classical arcade game. In order to do this, his project will examine eh importance of the gaming industry and the popularity of arcade game, then this project will discuss Artificial intelligence as it is a key feature to this game. Following this, the report will discuss the game design and examine why Unity and C# would be the best combination in the development of a classic arcade game. After this, the report will provide the technical explanation of the Board Class, the Ghost Class and the PacMan Class.

**Key terms:**

|  |  |
| --- | --- |
| GameObject | This refers to all entities in Unity, it is the base class. |
| GetComponent | :Using gameObject.GetComponent in the game will return the first component that is found, the order for recall is undefined. |
| Vector2 | This refers to the use of Vectors in a 2D setting, where there is only an X and Y axis for the sprites (?) to move along. |
| Transform.Position | This is the position property of a GameObject’s Transform, this is accessible in the Unity Editor and through scripts. To move a GameObject this value will need to be changed. |
| Transform.localPosition | The position of the transform which is relative to the parent transform. |
| Transfor.localRotation | Unity uses Quaternions to store rotations internally, the rotation of the transform relative can be used to transform the rotation of the parent. |
| IEnumerator | In Unity C# the IEnumerator is a return key that allows the game to be stopped and then recalled again from the same place. |
| Quaternion | These are compact ways of representing rotations and they can be easily inserted into other aspect |

Collected from the Unity User Manual [27]

**1.0 - Introduction**

The gaming industry is continually expanding in the modern era with the industry in 2022 being valued at over $300 billion. [1] The driving force behind the continued growth of gaming worldwide is directly linked with the new everyday discoveries and improvements in technology which have resulted in the quality and possibilities of gaming growing endlessly. Furthermore, this trend is almost guaranteed to continue for the foreseeable future as more innovative ideas, such as the Metaverse, continue to create opportunities for new and unique video games [2]. This revolution in the gaming world in recent years has resulted in the ability for individuals to make a livelihood from playing the games they love through new concepts such as social media platforms and competitive gaming (esports) [3]. These new developments have allowed the gaming industry to reach levels which could not be envisioned when the gaming industry first came into fruition in the 1960s. This paper aims to look at how the gaming industry was able to reach the heights it has elevated to in recent years by looking at one specific area in its history: the impactful success of arcade games during the 1980s.

Playing games in the playground at a young age or on a board game with your family have always served the purpose of relieving stress and increasing enjoyment. [4] When video games were invented in the 1960s, they served to emulate this and it succeeded. However, like any invention in technology, video games were originally a luxury as opposed to the modern era in which all our lives have in some way been influenced by video games as they surround us in our homes through consoles, computers and even in our mobile phones. [5] What really kickstarted the gaming industry was arguably the Atari Home console in the 1970s as well as its improved models. This came along with incredibly fun and immersive new arcade games such as Pong, Space Invaders and Pac-man. An arcade table tennis game known as Pong was the very first arcade game sensation to gain traction which allowed the sequel to the Atari, the Atari 2600, to become the first console to sell over a million units. This success was followed by another bestseller in Space Invaders. One could argue that Space Invaders is the most influential arcade game of all time as its success resulted in arcade machines being installed in many locations worldwide. [6] This leads us to the creation of what is undeniably the most well known arcade game of all time as well as the main focus of this paper, Pac-Man.

* 1. **- The significance of Pac-Man**

On the 22nd May 1980, Pac-Man was released on the Atari as well as many arcade gaming machines. Alternatively known as its original Japanese name ‘Puck Man’, Pac-Man was a maze action video game in which the player must help a character resembling a pizza with a slice removed around a maze. Pac-Man was significant in the growth of the gaming industry into what it is today as it was arguably the first example of gaming developers attempting to introduce women into the gaming industry. It was clear from the inception of video gaming that the industry had a significant male majority fanbase. Therefore, a young employee at Namco by the name of Toru Iwatani sought to develop a game that would appeal to women as he noticed a huge opportunity to attract players not just for Namco but also to the gaming industry in its entirety. He believed that women have stayed away from the gaming industry due to the sheer amount of violence and gore within video games at the time. This he believed resulted in an increase in a male player base whilst also deterring women from the industry. Therefore, his solution was to create a non-violent game that would appeal to women and couples. Essentially his aim was to develop a game that would be a catalyst and transform the gaming industry into a more family friendly nature. He ridiculously had an idea to involve eating as a major component of the game as he believed that a stereotypical hobby for women is their enjoyment of eating sweets and desserts. The then revolutionary RGB colour display also allowed him to use attractive colours as well as the ability to create cute characters which he also believed would help attract all types of players [7].

* 1. **– Gameplay**

The main purpose of the game is to feed Pac-Man all the dots along the maze whilst simultaneously avoiding four ghosts differing in colour known as Blinky, Pinky, Inky and Clyde. Eat all the dots and pass on to the next level with each level gaining in difficulty. If you are caught by any of the ghosts, you will lose a life. Around the edges of the maze are also powerups which will aid you by causing the ghosts to flee and also be eaten by Pac-Man. However, what really makes this arcade game stand out and become a major hit is how ahead of its time the game truly was. The Ai exhibited intelligence that far superseded other games during this time. An example of this is the movement patterns of the ghosts. Each ghost would have its own movement pattern in order to catch out the player. For instance, Blinky (the red ghost) would chase the player directly whereas Pinky (the pink ghost) and inky (the cyan ghost) would attempt to corner the player by appearing in front. Of course due to incapabilities of technology in the 1980s, the game was arguably not in its most perfect form which can be seen by the various changes which have occurred in the game’s sequels such as the Pac-Man character looking considerably different in recent titles. Nevertheless, the original Pac-man was undoubtedly revolutionary.

**1.3 - Why are people still playing them**

When people think about classic, or retro game design, they often think about the brutal difficulty of the games and likewise, when they think about modern game design, they often talk about modern games being “too easy” or “casual”. And while it is true that games have never been as accessible and as “easy” as they are today, the subject is not as simple as that, they often provide different types of challenges [17].

There tends to general conception that arcade games are relatively too hard, when compared to the a typical modern day game, games nowadays are being primarily designed to fit the modern day gamer. Many of which would prefer a more relaxed/casual experience than a frustrating one, However In my opinion the feeling of constantly being frustrated by an arcade game/ being stuck on a level gives a nostalgic sensation/feeling for most individuals, as this was something many people experienced during their childhood, therefore, recreating this experience was a crucial factor in the game design portion/stage.

**2.0 - Artificial Intelligence in Gaming**

The Turing test was developed by Alan Turing during the 1950s (check date and reference information). The purpose of the Turing test is to understand whether a computer will be able to function in a way that is similar to a human. This is done through rigorous testing that requires problem solving, Turing's method for evaluating Artificial intelligence was done through the “Imitation game”. The Turing test is a person game in which a computer communicates via text into manipulating a human being into thinking that the it is currently speaking to another person.

However, despite the seemingly impossibility of passing the Turing test Google Duplex has come close. For example, (insert year it happened) during testing an Artificial Intelligence device was able to book an appointment with a hairdresser, this was completed in the presence of additional people and was thought to be a revolutionary moment in AI voice technology. The AI stated that the purpose of the call was to book for a ‘client’, as well as negotiate a suitable time for the hairdresser and for the ‘client’. Part of the realism can be attributed to the use of ‘Mm-hmmm’ and ‘thanks’ which are informal speech, other key parts that affected the realism of the AI would be the tone of voice and the use of fillers such as ‘like’ and ‘um’. This is related to the Turing test as the discussed factors are a part of human behaviour and by factoring them into the research the developer is closer to making an AI that can pass the Turing test [21]

Turing, in one of his earlier papers it was proposed that machines will be able to have enough storage/capacity to make them capable of successfully completing the Turing Test. Turing’s hypothesis was that Artificial intelligence was already developed to a high enough level but the only aspect holding A.i from progressing was the capability of machines at the time.

**2.1 – Other Real World Applications of Artificial Intelligence-**

Artificial Intelligence refers to the process of how a machine tries to replicate the actions and the thought processes that a human might perform. For context, ‘intelligence’ is the ability that humans use on a daily basis to perform actions in real life. A.I tries to replicate this in certain instances. One instance would be Fraud checkers that the financial industry uses to detect any malfeasance in a user’s account, for example: if the type transaction made was not typical for the user, if the payment takes place in an irregular location, if the spending pattern of the user completely different to the daily life, the A.I program will detect this. Then it will proceed to send a notification to both the bank and user. [24]

**3.0 - Game design**

**3.1 - Sprite design**

All the Sprites for this game were designed in an application called GIMP, this stands for the GNU Image Manipulation Program, and it is a wildly successful multi-platform image editor. [15] This made the processor much quicker and allowed the ghost sprites to be more expressive in their different states.

**3.2 - Designing main menu**

The main menu of PacMan displays the starting aspects of the game, it is done using IEnumerator. The aim of the main menu is to allow the user to exit the game or star the game. At the same time, it reminds the user to ‘ger ready’ before the game starts.

**3.3 - Language of choice**

This section will explain why Unity and C# was preferred instead of using Java. This is an important section of discussion because the project was originally designed in Java. However, when conducting deeper background research in order to create an in-depth framework which could be used to plan and record the necessary tasks for project completion, Unity and C# seemed more suitable for the project requirements.

However, Java was originally chosen as the main language for this project due to the fact that all of the previous projects related to game design and game development were completed in Java. For example, a second-year project included the recreation of the classic arcade game ‘Space Invaders’ this game was primarily designed to be challenging, addicting and enjoyable for the consumer. The recreation of ‘Space Invaders’ would result in the necessary skill acquisition needed for further skill development. For example, Java illustrated how objects in classes would interact with another which is a key feature in Unity as each Class is assigned to a game object. Therefore, having a good understanding on the Parent/Child relationship helped create this project. Further interest in this project topic would come from initial field research: the playing of arcade games. As a result of this the arcade game that would be chosen for this project would be Pac-Man. This was finalised during September in order to present the project to the Supervisor: Dr Varasteh.

**3.3.1 - So Why Unity?**

Which made the decision of switching to Unity and C# a tough one as this is a language that I have no prior knowledge of

Unity has several features that benefit game design and game development. One of the key features of Unity is its flexibility, it seems to be able to do a little of everything. For example, Unity’s ability to adapt to any developer's needs makes it one of the most unique engines on the market. Unity primarily uses popular languages such as Javascript and C# which make it more efficient. Furthermore, it has impressive functionality as it can be used with all types of gaming consoles. [33]

**Implementing key mechanics**

**4.0 - Board Class**

This class function is essentially used to manipulate the user interface and present accurate information to the player such as the player's Score. This is the location where everything for the user interface is stored, a purpose of this class is to display key information on the game screen. This occurs through the three main processes which belong to this parent class.

**4.1 - Vectors**

Vectors are a significant factor in game development, this is because they can be used to move a character, choose a target in a first-person shooter or evening moving the cursor. In regard to this procet vectors will be used to allow the player to move the PacMan around the maze in order to consume the pellets. Additionally, the vectors will be used by the ghost to locate the position of the player in order to find the goal position.

Firsty, this section will provide a cursory overview as to what a vector is, then it will discuss how vectors are used in games to help programmers implement movement in an efficient way. A vector in game programming essentially describes two coordinates, these are X,Y,. The purpose of these vectors is to represent positions on the plane within the game. Vectors are mainly a mathematical function that considers both direction and magnitude; magnitude is the length of a vector; it can also be called the modulus of the vector. This game is in a 2D space where the Vectors can only have the X,Y positions. However in a 3D game there would be the X,Y,Z positions to consider.

//Looking at figure 1 compares the vector coordinate system to a cartesian plane

In order to calculate the distance between Vector A and Vector B a simple calculation is used

|AB|= (x2-x1)^2 +(y2-y1)^2) This is the primary way that movement is done in this project.

(X,Y)

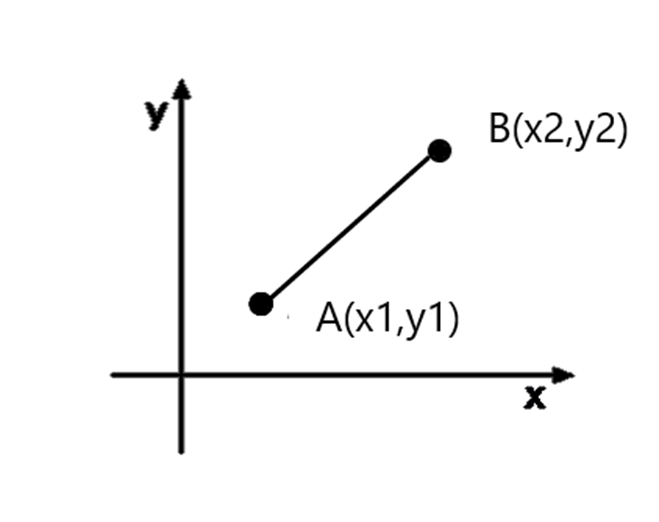


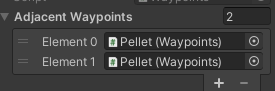
Figure (1) is a graph to compare the vector coordinate system to a cartesian plane

**4.2 - Creating Objects within the Map**

The board class creates an array of objects and uses a foreach loop which goes through each object in this loop, this transforms the vector2 position of each object and stores it into the array, using Unity’s .name and. functions we can specify what is wanted from this array. This is to essentially transform the objects in board into vector positions so it can be called upon from other classes.

**4.3 - Waypoint movement**

This class helps build the foundations of the game, it has a fundamental purpose which is to help control the movement of all objects that are within the game, [18]. It starts by creating waypoints at each intersection of the game, this is the position where PacMan is able change directions, this is done through the class by initialising two arrays, the first one is labelled ‘Adjacent Waypoints’. The other is defining a vector 2 array of the possible vector locations that the player can take see figure (2), the adjacent waypoint array was set manually in the inspector where each waypoint was assigned a corresponding waypoint see figure(a)as each intersection was planned during the game design phase. Once all these were created, movement scripts that use these waypoints were built to allow players to move efficiently without realising they have constricted movement in a sense. Examine appendix two to see how waypoints are implemented within the maze.



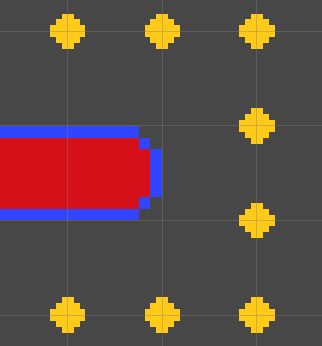


Figure (2) top right is the one to highlight to show the corresponding waypoints

Figure (3) is to indicate to the waypoint.

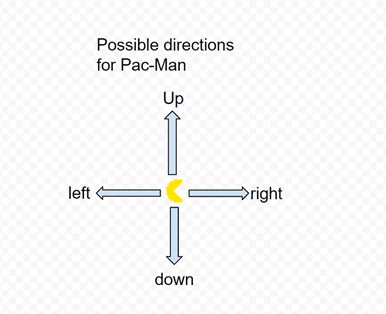
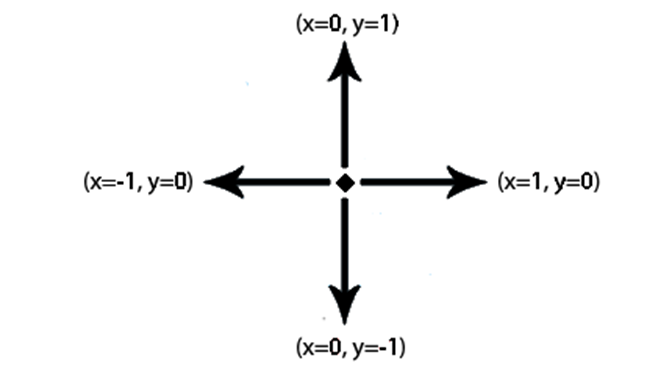
****

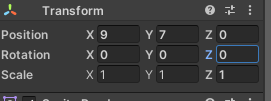
Figure (4) left is a graph to show the four different directions that the ghosts and PacMan can move in using vectors. Figure (5) is a diagram to show the directions PacMan can move in.

**5.0 - Pacman Class**

**5.1 - Setting the parent class**

The PacMan class is the parent class that uses several methods to manipulate the movement of the PacMan GameObject. It stores the two possible PacMan animations that were created. These animations are the: PacMan eat animation and the PacMan death animation. These animations are controlled by Unity's animation controller and the Setupboard Class. Only one PacMan eat animation clip had to be created, this was due to the fact that the rotation of the game object was done using Unity's built-in function of transform local rotation and local scale. We use the transform local rotation Quaternion.Euler to change the sprite of PacMan

Euler angles are the angles of rotation that are seen within a 3D coordinate frame. This rotation is depicted as a ‘matrix of trigonometric function’ of angles. In comparison to Euler angles, quaternions which are centred around algebraic structures, are more difficult to understand. However, the use of quaternions instead of Euler angles is common, this is because quaternions offer better stability and can therefore be used with increased efficiency. [19]



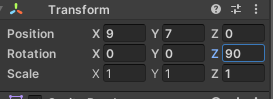
****Figure (6a) is PacMan facing right, figure (6b) shows that no values have been changed to his rotation

Figure (7a) is showing what happens when you change the z value in the transform component, it rotates PacMan up (figure 7b), these rotations are defined in the PacMan animation class.

**5.2 - Moving PacMan around the maze**

Pacman can only move within the realms of the maze, in order to make PacMan move there are several different methods that can help the PacMan object to move around the game, The first one is the PacMan User Controller, this method will make use of Unity’s built in Input Class, where it uses a method called GetKeyDown to receive input from the player. For instance, in this project’s game, if the player was to press the down key, it would then calls a method called TransformPacMan. This will always be true since the variable defined in PacMan’s base class is set to zero, therefore, whenever a user presses a key the method will check to see if movement is possible in that direction. This occurs through a selection of waypoint detection classes which are set up in PacMan’s game object; if the user's input has a target waypoint it then sets the player to move in that direction. Additionally, there is also a class that is labelled ChecktoSeeifMovementIsPossible which essentially constricts the ghosts and PacMan to stay within the realms of the maze.

**5.4 - Eating pellets**

This is the Eat Class, in this class the pellets are consumed and the audio is stored. There are three functions. The first function is to manipulate the audio, there is an audio file attached to the Inspector it checks if the Boolean value is true, if it is true the audio is played. This method is then later called by the ‘eat function’ when a pellet is consumed this audio clip will play. The second function is the aforementioned ‘eat function’ in this function its purpose is to call upon the frame class to check if the block that is currently in the grid has been eaten, this is done through the Boolean value, if the tile component has not been yet consumed by it has Boolean value of false, once the player enters the proximity of the pellet or has eaten the pellet the class calls the sprite render to disable the sprite, this is in order to keep the waypoints active at this point, the score is also incremented by 10.

If the player eats a power pellet, this will begin ‘scaredmode’ which is later defined in the class. The purpose of this is to check the state of the game, if it is not in the eaten state then the timer is reset, this then increments according to the ghost parent class where all timers are initialised, essentially checking to see if the complement in the current block is known.

**6.0 - Ghost Class**

**6.1 - Introducing the ghosts**

There are four ghosts that make up the enemies, the first is Blinky, the red ghost. It is arguably the most well-known ghost as it is the most hostile ghost in the game, this is because it will relentlessly chase Pacman during most stages of the game [50] At the beginning of the game Blinky is placed on a tile just above the ghost house. This will be Blinky’s starting position unless he is eaten by PacMan, in that case he will transition into the “eaten state” where he will temporarily return to the ghost house.

Blinky has a fairly simple AI that makes him chase PacMan, the AI works by locating PacMan’s position, then it draws a vector. Once the shortest distance is established: through the ‘GoalPositionOfGhost’ and will target the tile position of the shortest route. Blinky will then continue to chase PacMan until PacMan consumes a power pellet, then Blinky will enter the scared state. If PacMan, after he has consumed the pellet and within the 10 second timer, consumes Blinky, Blinky will enter the eaten state and return to the ghost house where he will immediately vacate.

The other three ghosts are: Pinky (Pink Ghost), Inky (Blue Ghost) and Clyde (Orange Ghost). In comparison to the Red Ghost, they are less aggressive, they all begin in the ghost house and leave one at a time. The first ghost to start chasing PacMan is the Pink ghost who waits five seconds after the Red ghost starts chasing, then the Blue Ghost will follow five seconds after the Pink ghost and the last ghost to leave will be the Orange Ghost who waits a sum total of twenty seconds to leave the house once the game begins.

**6.2 - Setting up the Parent Ghost Class**

The ghost parent class is how all the initialisers, timers, variables, sprite storage and the waypoints of all the ghosts are set up. Alongside this an enumerator class was created called enemy states, this represents four different states of the ghosts in the, there is also another enumerator initialised called ghost colour which represents the four different types of enemies in the game, red, pink, blue, and orange.

In the unity start methods there are several reference scripts that the ghost class uses, to help the ghosts manoeuvre and for the ghost enemies to do their tasks, which is to be constantly hunting down PacMan. In unities’ update method which gets called once per frame there is an if statement to check if the ghost is able to move or if the Boolean value is true then the ghost are able to change states and are allowed to move.

Through the use of the enumerators that were previously defined in the class there is another public Vector2 class that makes use of four separate scripts that were created to manipulate the artificial intelligence/movement pattern of four different types of ghosts that are in our game. This method uses a switch case to manipulate the instances of the ghosts, for instance if the GameObject has the enumerator definition of red then this method will call upon the Red ghosts pathing algorithm to find/hunt PacMan down.

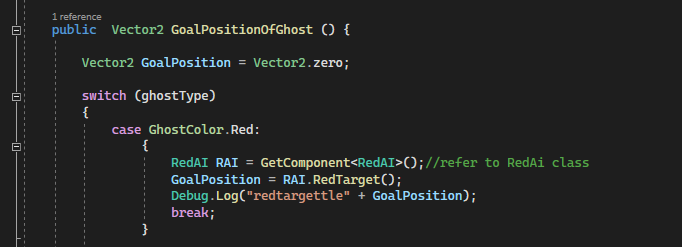


Figure (8): This screenshot of the code used defines the objective waypoint for each individual ghost.

**6.3 - The Ghost House**

The Ghost house is located at the centre of the maze, and it is the place in which the ghosts

will leave from. At the beginning of level three of the four ghosts will be inside the ghost house, the only ghost that will be outside the confines of the house will be the Red ghost. The other three ghosts will be allowed to leave the Ghost house one at a time every five seconds. Within the Ghost house there are two waypoints labelled “ghost house”, these points are a public Boolean, this is required because they will need to be accessed by other components of the game such as the timer that releases the ghosts.

The timer that controls the speed at which the ghosts leave the Ghost house is incremented in Time.deltatime. These timers are initialised in the ghost class as this would be the main class that stores all the necessary variables needed to make the game function. There are four individual void functions that is individualistic to each ghost, for instance in figure (8), there is an if statement that refers to the enumerator in the ghost class that inspects to see if the ghost is the type orange and if it is in the ghost house, if this statement is true it the Boolean value of Ghost house will change to false be released in from the Ghost house. Furthermore, in order to prevent the ghosts from ignoring the confines of the Ghost house walls, the only movement allowed inside the ghost house are waypoints that are adjacent to it.

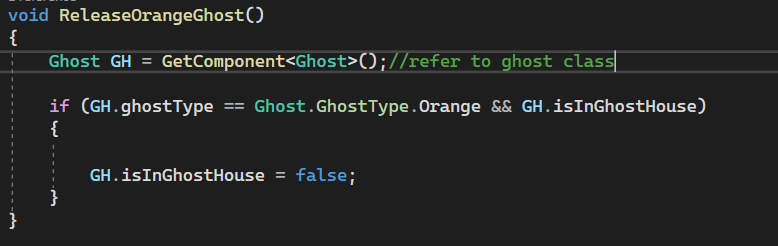
Looking at figure (9) the method Release ghost is attached to all the ghosts as it manipulates the order in which the ghosts will be released. Secondly a public Boolean variable is instantized in the ghost class of ‘isinghosthouse’.

Figure (9) is a screenshot of the code needed to implement the order in which the ghosts leave eth Ghost house.

Ghost controller is when a ghost is in an eaten state, the purpose of this class checks to see if the ghost in the eaten state is in the ghost house, this is done by locating the tile from the block location function. If the tile is not null we call upon the frame class to see if that frame has a Boolean value of true, this will then set the current speed of the eaten ghost back into its normal chase mode speed, it will then only allow the ghost to travel up, it will also transition the ghost back into its normal animation through the use of a reference of the ‘AnimateInRealTimeClass’.

**6,4 - Red ghosts Artificial Intelligence class:**

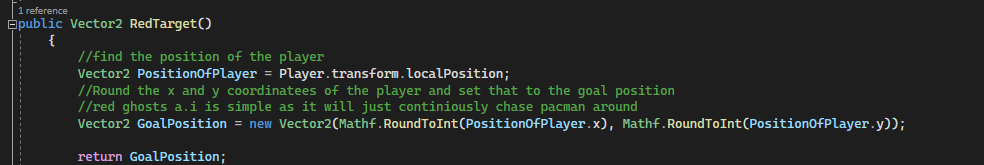


Figure 10 shows the AI used to implement the Red ghost.

The Red ghost’s artificial intelligence makes use of the Vector2 class in order to detect what position the Player is currently at. It then rounds the position of the player, who is at an x and y coordinate to the nearest integer. This method is then called by the goal position class, this is located within the parent Ghost class. This ghost will then continue to follow the player until the state changes. Red’s pathfinding method is an integral part of the game, as he is attempting to find the shortest pathway to PacMan [32] this is because the others ghosts also utilise Red’s position in order to make their own paths and movements.

**6.5 - Blue ghosts Artificial intelligence class:**

Blue’s ghost class is the most unique, in comparison to the other ghosts. This is because Blue’s artificial intelligence uses the Red ghost location to determine its own target position. Firstly, it finds the current vector space that the PacMan player is in, then it identifies the direction in which the current player is going in, this is defined as the ‘goal position’. From here Blue proceeds to find the Red ghosts’ current position, then using Unity’s inbuilt distance class he calculates how far away the Red ghost is from the goal position. This value is then multiplied by 2 and this is then referred to as a ‘float’.

This class then redefines the goal position of the Blue ghost by adding the Red ghosts current x and y position to the Boolean value which was previously defined. This method adds a very random component to the Blue ghost because he does not follow any predictable pattern or way that would be expected of an arcade enemy, therefore this could be seen as an ‘ambush algorithm’.

**6.6 - Oranges ghosts Artificial intelligence class:**

In comparison to the other ghosts, the Orange ghost has a more simplified movement pattern. The ghost attacks PacMan by working out the player's current position, then the Orange ghost looks at his own position. If the distance between PacMan and the Orange Ghost is greater than 8, the Orange ghost will begin the chase state by following PacMan. However, once the player it targets the current player, if the player is within 8 tiles it will target its scatter waypoint.

**6.7 - Pink ghosts Artificial intelligence class:**

The Pink ghost will use Vector2 to identify the position of the player, this is the local position. Then the Pink ghost will add 4 to PacMan’s position in the direction he is moving in, this will become the goal position.

**6.8 - Ghost Animation Class**

The ghost animation class is responsible for handling all the possible animations the ghosts can have, as previously mentioned the ghosts can have up to four different states. This means there needs to be four different separate animation clips therefore/which resulted in separate animation sets/clips being produced. The first state is the ‘regular state’, in this state the ghosts will be animated in order to look like they are floating and running around the maze, this required several different sprites in order to successfully represent the movement across the grid, the aim of the animation was to make the legs move forwards and backwards by changing between the designed sprites. The second animation was designed to show that the ghost was in the ‘scared state’, this state is activated once PacMan consumes a power pellet. This is a necessary animation as it indicates to the player that all ghosts who are currently roaming the maze are consumable. While they are in the scared mode, and before they have been consumed there is an additional animation. The purpose of this animation is to indicate to the player that ghosts are transitioning from a scared state back into their previous state which could be either chase or scatter. The final function of this class will be to change the movement of the ghost's eyes. This is because when the ghosts have been devoured the selected sprite will be and must return to the ghost house.

In total there are four separate animations, one for each of the ghosts, because the ghost can change directions there were 32 different sprites that were created, which is broken down into eight sprites per ghost. The difference between the sprites is that one sprite would be that the sprite for the ghost when they headed in the left direction, additionally the sprite would have two legs and the other sprite would contain 3 legs, by alternating between the sprites in the animation it would look like the ghost is running.

The key component in manipulating these animations is the changing between the different frames of the sprites. This happens using ‘if statements’ and the ‘GetComponent’ Method. This class makes use of Unity’s base class of animator and sprite renderer, the animator class calls upon the runtime animator controller which is the in game representation of AnimatorController that helps to choose what type of animation to play. [38]

**6.9 - Finite State Machines**

A finite state machine (FSM) is a model that depicts a device in which there is a finite number of states, alongside this the device can only be in one state at a time and changing between states relies on an input from outside of the model. The simplicity of the FSM makes it an attractive way to make characters in the game appear intelligent. [9] There are two types of Finite state machines, deterministic FSM and a non-deterministic FSM [14]

Finite state machines are really popular in many games, because it is a good way to design how enemies act, especially in boss fights. For instance, if the health of a boss were to decrease below a certain threshold change his state to an enraged state/ this is how many games design boss levels that have several different mechanics. It is also how PacMan is designed as It has three possible states.

Another example of using FSMs in a video game is through implementing a ‘non-player character’ to have a certain reaction based on a particular input. For example, it may not be possible to have a certain interaction with an NPC until certain conditions or objects have been met. [10] Additionally, using a FSM method can result in more immersive gameplay through interactions with an NPC that acts intelligently can allow a more tailored experience, in one instance if the player attacks and NPC their interaction occurs, after the player leaves the location the interaction can be reset, allowing the player to have the same experience again as long as they are implementing the input (proximity to npc) which will change the from inactive state to an attack state [11] this can also make video games more challenging and therefore more rewarding, this is because, by fighting an NPC it removes the chance of human error, the NPC may require a specific input in order to move from an active state to a deactivated state. [13]

**6.9.1 - Ways of implementing FSM into the project:**

Examining other projects implementation of finite state machines into games that are similar to pacman it is shown how the finite state machine can either be simplistic or extremely developed [12]. In order to test the effectiveness of a FSM it is important to examine the following sections: state identification, state verification, fault detection [16] this can be applied to the testing of this project's FSM.

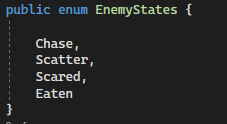
**6.9.2 -How Finite State Machines are programmed into the Game**

Figure (11): Creating an enumerator/finite state machine for all the possible modes.

**6.10 - The Ghost States Class:**

There are two public methods that help manipulate the states of the Ghosts, these are called ChangeStateOfGhosts() and ImplementChange, Changestate makes use of the parent ghost classes to determine what current mode the ghosts are currently in. The ghost class also has several timers defined, these represent/determine how long a ghost will stay in a particular state for (see figure x) There is a Statechange timer that is initialised to zero in the ghost class, this is then incremented at the start of the ChangeStateOfGhost method in Time.deltatime. Another variable that is crucial to this class is the StateIncrementer, this helps to determine which stage the ghosts are in, this is because there are several different instances of the possible states.

There are then several if statement checks to see if the statechange timer exceeds the timers initialised in the parent class. For instance, looking at figure (13) if the initial timer exceeds the scatter timer then the state incrementor will be called, this will result in the game transitioning the ghosts into the next natural state defined, this will then reset the base state timer.

Figure (12) is an example of if statement to change States of ghosts.

Figure (13) is a screengrab to show the scatter and chase mode timers.

This class also helps us implement a classic feature in the arcade game, which is to make the ghosts blink just as they are transitioning from the a scared state back into the last state they were in, this is to help inform the player that the ghosts will now be able to kill PacMan, this is also done through a basic timer.

There is another public method defined in this class. This is called implementchangeOfState, the purpose of this class is to manipulate the ghost's speed according to what state it is in (this is illustrated in figure ©). This figure will demonstrate what will occur when PacMan eats a power pellet. This method will get called as the ghosts will transition into a scared state which then alters their current movement speed to a much slower rate. This will result in the function then storing the current enemy speed into the field defined as StoreMoveSpeed. This variable is crucial for when it is necessary to manipulate the ghost's speed back into its normal state of either chase or scatter. During these states the speed is set at a base 5.9f.

These two methods are essentially how the game transitions from one mode of the game to another, as ImplementChangeOFState is called by ChangeStateOfGhost at every if statement if the loop is true, this will then change the state of the ghosts according to the if loop.

At the start of every game/death the ghosts will be in a scatter state, they will then transition to a chase state after the first scatter timer, this cycle is then repeated four times as defined in the parent ghost class. After four increments of these timers the ghost will be in permanent chase mode, this is assuming that the player has not eaten a power pellet and changed the current increment timer.

**6.11 - The Ghost Direction Decision Class**

The ghost direction decision class contains two functions. The first function is the GhostDecisionMethod(), this method's job is to alter the goal position of the ghosts. To do this it calls upon the parent class Ghost so it can find the possible states the ghosts are in. To perform this action the method uses a switch case method.

The second function sets up two arrays, one that calls upon the waypoints class. The other is for all the possible directions that the ghost can take. These arrays are then used to control how ghosts move along with other functions. The method that is used is a loop. It goes over to the ghost’s current waypoint to detect where it could potentially move. Since in the original PacMan the ghosts are not able to move in reverse directions, this feature is implemented in line 57. The method then checks to see if the current state of the ghost is in any other state besides the eaten one, it then uses a reference script to find the coordinates of the location of the ghost, succeeding this it then sets the ghosts current waypoint to this position.

This method also prevents PacMan’s early death, this is because the method prevents the ghosts from taking the shortest path to PacMan in the instance he is directly below the house. This is because the waypoints that are placed in the ghost house do not have any corresponding waypoints outside of the Ghost house that will allow the movement of the Ghost from inside the house to below the house. This is because the ghost are set to find the shortest pathway between their position and PacMan [25]

Furthermore, there are three ‘if’ and ‘else’ statements which are used in this method, they essentially increment the waypoints into an array and also store the possible vector positions that the ghost can take, at the end of each statement the waypoint incrementer is increased if the loop is true, since there can only be four possible waypoints and four possible direction this only will increment to the value of four. There is then an if state statement to check if the length of the waypoint is equivalent to one, meaning that only one waypoint was found, this is then set to the ghost classes possible movement decision, there is also an if statement to check if there is more than one waypoint found, it will then find the quickest path that will take the ghost to its goal position [26]

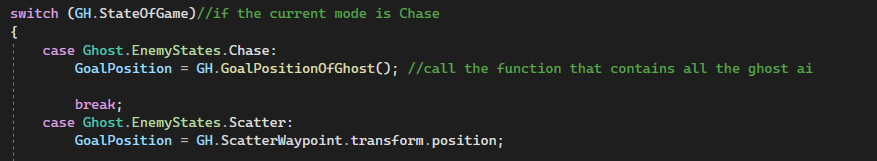
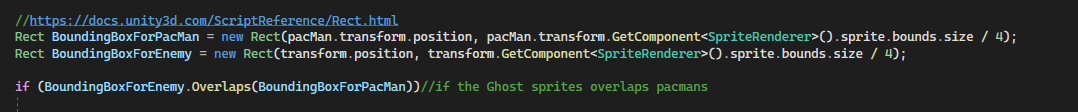
To summarise, this class essentially looks for all the neighbouring nodes and adds them to an array, as seen in figure (14).

Figure (14) is a screenshot to show the code of how the ghosts change direction.

**6.12 - How Collision Works in this Game.**

If PacMan is more than halfway through he will then be legally within the tile, this is the same approach for the consume method. For example, if the ghost reaches the legal tile within the proximity of PacMan, PacMan will be consumed. For this to apply the ghost also has to be more than halfway into the tile for it to be classed as legal.

The way collision works is similar to how box colliders work [28] it uses Unity’s Rectangle built in class to detect collision [29]. For example, the rectangle class is attached to both PacMan and Ghosts, through the use of an in-built function called Overlaps(), this can detect if either component enters the legal space of each other. In cases where this occurs, the game can implement methods to manipulate two different outcomes. The first would be that either the ghost is going to be consumed because it is a scared state which will then call upon the EnemyEaten method. The second option will be that the PacMan will trigger the death method.

Figure 15 is a calculation to show the use on an IEnumerator.

This method essentially defines what the game will do if the ghost is in a scared state and has been consumed by the player, it manipulates the key components of the ghost class such as the state and animation change. It also helps add a feature that was included in the classic game, this was where the player would be rewarded with more points depending on how many consecutive ghosts he can eat. This is done through a basic calculation (see figure 15) and through the use of an IEnumerator.

The way in which Ghost eaten score is displayed is through the use of a built in unity feature called WorldToViewPoint, which essentially converts the ghosts local game position to the canvas position as they both have different positions on the game, since all the user interface is designed on the canvas there is a need to display the text where the ghost was consumed as it would be confusing to the user if they were to complete a task and not be notified of its completion. [30] Figure (16) shows how the players score is calculated and displayed.

**7.0 - The Board /Manager Game Object  
7.1 - The Purpose of Board Class/”Manager”**

This Class is the primary way in which the text and images on our game screen are controlled, this information is required as it provides the player with the necessary information, for example their current score and the number of lives they have left, this is done through the method LivesCheck(). One of the reasons this works is because the method is attached to the “Manager”.

There are several key components that go into the processing of the important aspects within this game. Evidence of this, is that this Class will program what will happen at the crucial instances of a game. For example, at the start of the game there is text that appears on the screen to notify the player that the game is about to start, this is all processed through the use of Coroutines. A coroutine allows the developer to spread tasks over several different frames. In Unity, where this game is developed, a coroutine can pause an execution and return the control back to Unity, however it can resume again where it was paused in the following frame.[27]

At the start of every game there is a class defined as InitialStageOfGameScreen, this class has three different crucial functions that rely on each other to display the required text and sprites at key times. This class can also stop for movements at times, for example when an animation/text needs to appear. This methodology is similar to aspects of this classes that are attached to the “Manager” game object.

The first stage of the methodology begins by disabling the ghost and players sprites, it also disables the ability for the player to move as this would be seen as glitchy. This state occurs for about 1.0f(check), once the time is up the class then re-enables the sprites at stage 2. This stage occurs for 2.f, following this it then proceeds to the next stage through a coroutine of stage 3 , this is where it renews the player PacMan's ability to move and it also disables the “Get Ready'' text.

An additional process that needs to be controlled by the “Manager”GameObject and what processes occur when PacMan dies. This is relatively similar to the restart script; it has three stages that all correspond to each other. The first stage only gets called upon if a ghost manages to consume PacMan, the BoardSetUp class then checks the boolean value of the variable isObjectDead, which is initialised to false, Stage 1 then disables any game objects from moving, it also disables any animation that will be occurring. it sets the boolean value of the field isObjectDead to true. The last step of this stage is to start the coroutine for stage 2, which disables all sprites that are currently active in the game screen, this process then starts stage 3 where the main object of this IEnumerator is to find the game object “PacMan” and start its Death animation clip, it will also play the PacMan death audio clip.

The BoardSetUp class also controls how a player is able to progress onto the next level, this is done by the game scanning how many active pellets are on the grid/map, if the total number of pellets is equivalent to the number of pellets eaten by the player it then starts the process of the player completing the level, this is also done through an enumerator

Where it stops/ all game components of the game that are currently active.

**8.0 - Testing**

The main way in which testing took place is through ‘debug.log’, the purpose of this application is to identify any errors or bugs in the game. Placing the debug.log at instances in order to understand if there is an issue by examining a significant amount of data. In order to check additional issues, the game was tested by a group of participants as they would likely all play the game differently which would showcase any issues. After they completed the game there were given a blank form to fill in (see appendix 5) For example, those who played the game before realised that the PacMan death animation did not look right. This was a result of a bug in the animation which prevented the sprites from changing when receiving the input.

**9.0 - Project Management**

**9.1 - Methodology**

Jira software was initially designed as a tracker for bugs and other issues, but it has found its success in being used in project management. It allows the user to develop a roadmap through the use of a Kanban Board, which will guide them towards a clear end goal. It's clear design allows you to see projects that are in progress, on your to-do list or that have been completed. It is extremely helpful as it lets you search for specific items, this is necessary as many projects have several steps which could lead to overcrowding. Additionally, tasks can be assigned expected times of competition, this is important as it ensures that the members of the project remain consistent with their work output.

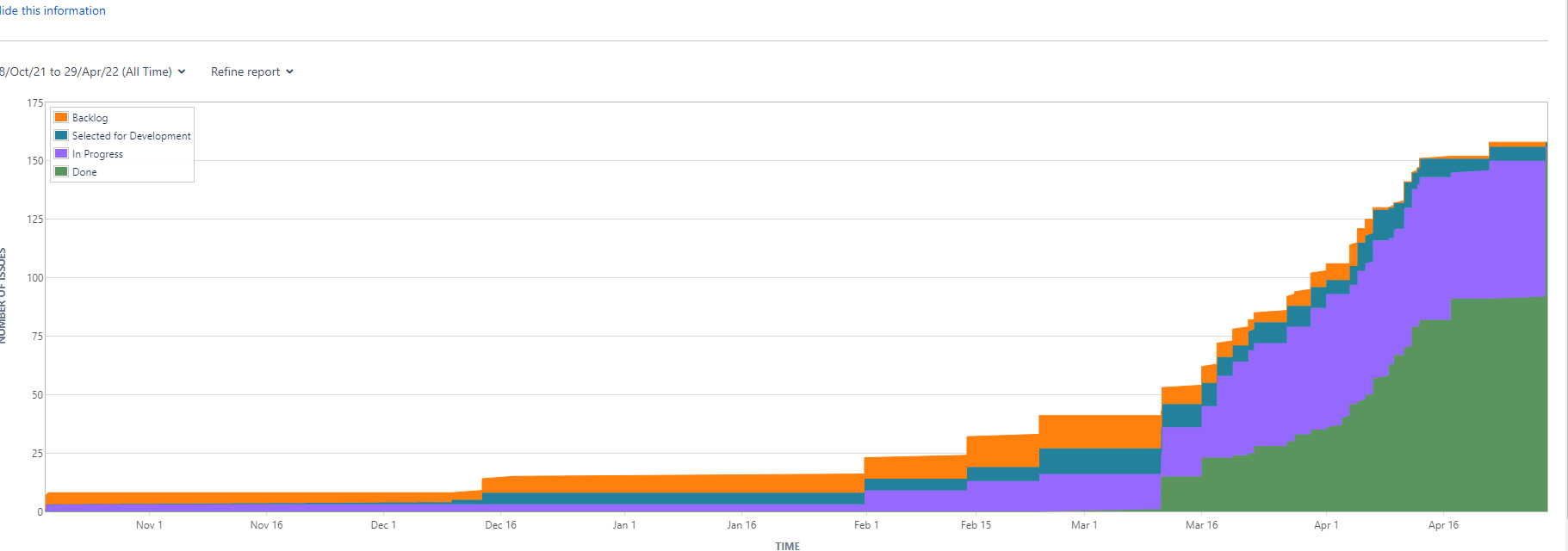


Figure 16 is a Cumulative Flow Diagram

Control

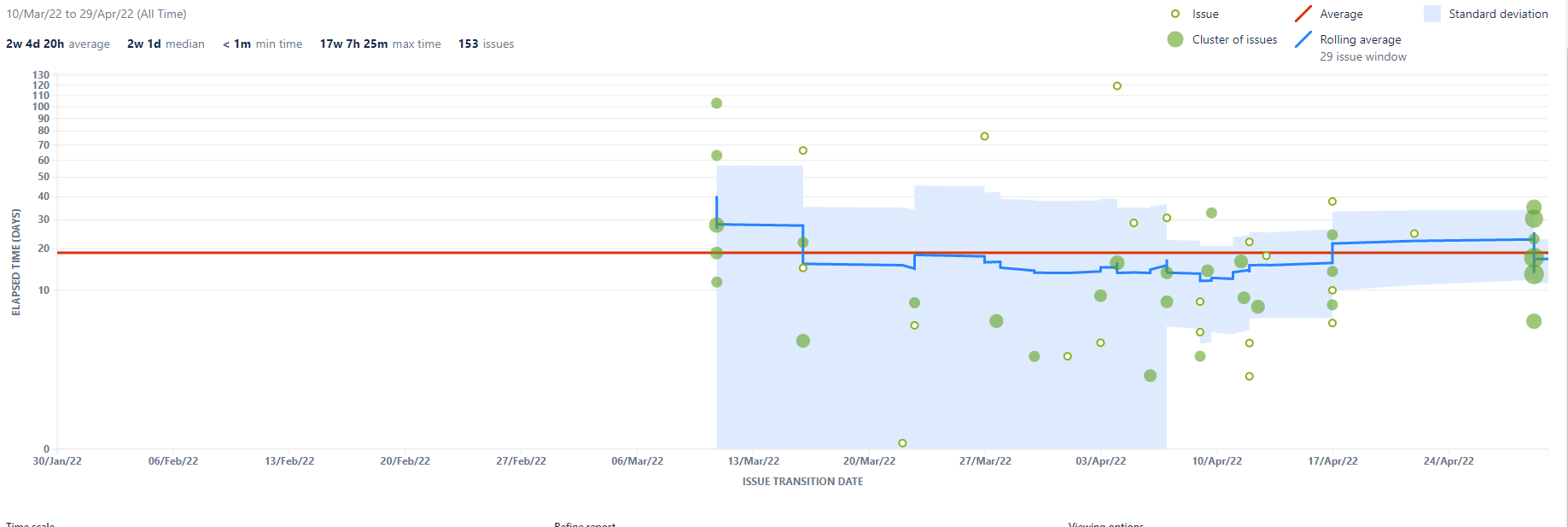


Figure 17 is a control flow chart.

This project made use of the Kanban board in order to organise tasks that had been completed and to plan and allocate time for tasks that needed to be completed. This is a tool that I am familiar with through previous work, it has been especially helpful in teamwork projects as it has an ‘assign to other’ feature. The Kanban board was an obvious choice as its segmentation of larger tasks into smaller tasks more manageable. [23]

**10 - Conclusion**

This project, while often very difficult, has been extremely rewarding and has allowed further skill development n regards to games and the use of Unity. However, as seen on the Kanban board there were more parts o his game that could not be included despite the time it took to plan and execute key factors there was not enough time to do the game justice therefore this project, while including some different features, has paid homage to the popular video arcade game Pac-Man.

**Bibliography**

[1] Raza, F., 2022. How Much Will The Gaming Industry Be Worth In 2022?. [online] Influencive. Available at: <https://www.influencive.com/how-much-will-the-gaming-industry-be-worth-in-2022/>

[2] Dionisio, J. D. N., III, W. G. B., & Gilbert, R. 2013. 3D virtual worlds and the metaverse: Current status and future possibilities. ACM Computing Surveys, 45(3), 1-38.

[3] Törhönen, M., Sjöblom, M., Vahlo, J., & Hamari, J. 2020. View, play and pay?–The relationship between consumption of gaming video content and video game playing and buying. In Proceedings of the 53rd Hawaii International Conference on System Sciences. Hawaii International Conference on System Sciences.

[4]Russoniello, C. V., O’Brien, K., & Parks, J. M. 2009. The effectiveness of casual video games in improving mood and decreasing stress. Journal of CyberTherapy & Rehabilitation, 2(1), 53-66.

[5] Prot, S., Anderson, C. A., Gentile, D. A., Brown, S. C., & Swing, E. L. 2014. The positive and negative effects of video game play. Media and the well-being of children and adolescents, 109, 2010-2014.

[6] Picard, M. 2013. The foundation of geemu: A brief history of early Japanese video games. Game Studies, 13(2).

[7] [Purchese](https://www.eurogamer.net/authors/robert-purchese), R. 2010. Iwatani: Pac-Man was made for women. Eurogamer. Available at: <https://www.eurogamer.net/iwatani-pac-man-was-made-for-women>

[9]Buckland, M. (2005). *Programming game AI by example*. Jones & Bartlett Learning.

[10] Syahputra, M. F., Arippa, A., Rahmat, R. F., & Andayani, U. (2019, June). Historical theme game using finite state machine for actor behaviour. In Journal of Physics: Conference Series (Vol. 1235, No. 1, p. 012122). IOP Publishing.

[11] Fathoni, K., Hakkun, R. Y., & Nurhadi, H. A. T. (2020, July). Finite State Machines for Building Believable Non-Playable Character in the Game of Khalid ibn Al-Walid. In *Journal of Physics: Conference Series* (Vol. 1577, No. 1, p. 012018). IOP Publishing.

[12] Gallagher, M., & Ryan, A. (2003, December). Learning to play Pac-Man: An evolutionary, rule-based approach. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* (Vol. 4, pp. 2462-2469). IEEE. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1299397&casa_token=KHrBbzmCflMAAAAA:jUzbTy-b7DZF-9QrwCrfDMvHywDxpTNflsFzRT4aLpnodtEfOJAWqcPtpuCrNiX6k2d9Ww&tag=1>

[13] Mora, A. M., Merelo, J. J., García-Sánchez, P., Castillo, P. A., Rodríguez-Domingo, M. S., & Hidalgo-Bermúdez, R. M. (2014). Creating autonomous agents for playing Super Mario Bros game by means of evolutionary finite state machines. *Evolutionary Intelligence*, *6*(4), 205-218.

[14] Soon, G. K., Guan, T. T., On, C. K., Alfred, R., & Anthony, P. (2013, November). A comparison on the performance of crossover techniques in video game. In *2013 IEEE international conference on control system, computing and engineering* (pp. 493-498). IEEE.

[15] Comber, O., Motschnig, R., Mayer, H., & Haselberger, D. 2019, April. Engaging students in computer science education through game development with unity. In *2019 ieee global engineering education conference (educon)* (pp. 199-205). IEEE.

[16]Yannakakis, M. (1991, January). Testing finite state machines. In Proceedings of the twenty-third annual ACM symposium on Theory of computing (pp. 476-485).

[17]Hyttinen, T. (2018). An analysis of modern retrogames: project: Queen's Meadow.[<https://www.theseus.fi/bitstream/handle/10024/150275/Hyttinen_Teppo.pdf?sequence=1> ]

[18] Harsani, P., Mulyana, I., & Zakaria, D. (2018, March). Fuzzy logic and A\* algorithm implementation on goat foraging games. In *IOP Conference Series: Materials Science and Engineering* (Vol. 332, No. 1, p. 012054). IOP Publishing. <https://iopscience.iop.org/article/10.1088/1757-899X/332/1/012054/pdf>

[19] Ben-Ari, M. 2014. A tutorial on euler angles and quaternions. *Weizmann Institute of Science, Israel*,2*4*.<https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf>

[20] Habgood, M. J., Moore, D., Wilson, D., & Alapont, S. (2018, March). Rapid, continuous movement between nodes as an accessible virtual reality locomotion technique. In 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) (pp. 371-378). IEEE.

<http://shura.shu.ac.uk/18594/1/Habgood-RapidContinuousMovementBetweenNodes%28AM%29.pdf>

[21] O'Leary, D. E. 2019. GOOGLE'S Duplex: Pretending to be human. *Intelligent Systems in Accounting, Finance and Management*, *26*(1), 46-53.

**[23]** Mishra, A., & Mishra, D. (2013). Software project management tools: a brief comparative view. ACM SIGSOFT Software Engineering Notes, 38(3), 1-4.

[24] Kaur, D., Sahdev, S. L., Sharma, D., & Siddiqui, L. (2020). Banking 4.0:‘The Influence of Artificial Intelligence on the Banking Industry & How AI Is Changing the Face of Modern Day Banks’. *International Journal of Management*, *11*(6).

[25] Hu, J., gen Wan, W., & Yu, X. (2012, July). A pathfinding algorithm in real-time strategy game based on Unity3D. In *2012 International Conference on Audio, Language and Image Processing* (pp. 1159-1162). IEEE.<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6376792>

[26] Patek, S. D. (1997). *Stochastic and shortest path games: theory and algorithms* (Doctoral dissertation, Massachusetts Institute of Technology). (see folder)

[27] Unity, 2021, *Unity User Manual 2021.3 (LTS)* Unity Documentation

[28] Calabrese, D. (2014). *Unity 2D game development*. Packt publishing.

[29] Unity, 2021, Script Reference. Unity Documentation. Available at: https://docs.unity3d.com/ScriptReference/Rect.html

[30] Unity, 2017, Camera.WorldToViewportPoint. Unity Documentation. Available at: https://docs.unity3d.com/2017.4/Documentation/ScriptReference/Camera.WorldToViewportPoint.html

[31] Svensson, J., & Johansson, S. J. (2012, September). Influence Map-based controllers for Ms. PacMan and the ghosts. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)* (pp. 257-264). IEEE. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6374164>

[32] Golden, B. (1976). Shortest-path algorithms: A comparison. *Operations Research*, *24*(6), 1164-1168.<https://pubsonline.informs.org/doi/pdf/10.1287/opre.24.6.1164>

[33] Glownight Games, 2020, Reasons To Choose Unity 3D For Mobile Game Development Available at: [<https://glownight-game.medium.com/reasons-to-choose-unity-3d-for-mobile-game-development-cb938bd47596>]

[34]Mathew, G. E. (2015). Direction based heuristic for pathfinding in video games. Procedia Computer Science, 47, 262-271.

[35] Johnson, D. B. (1973). A note on Dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, *20*(3), 385-388. <https://dl.acm.org/doi/pdf/10.1145/321765.321768>

[36] Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. Operations research, 17(3), 395-412. <https://www.jstor.org/stable/pdf/168375.pdf>

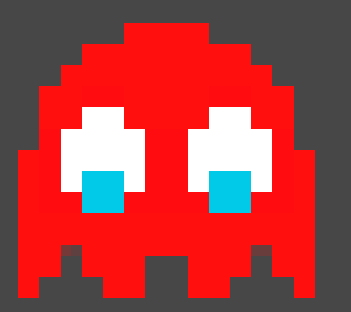
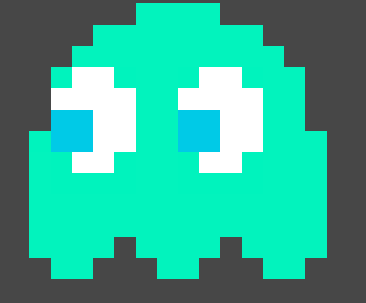
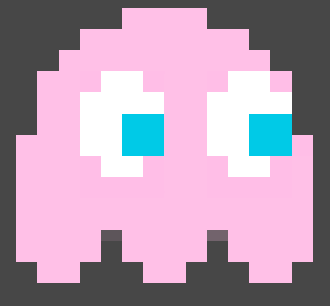
[37]Papadimitriou, C. H., & Yannakakis, M. (1991). Shortest paths without a map.

*Theoretical Computer Science*, *84*(1), 127-150.

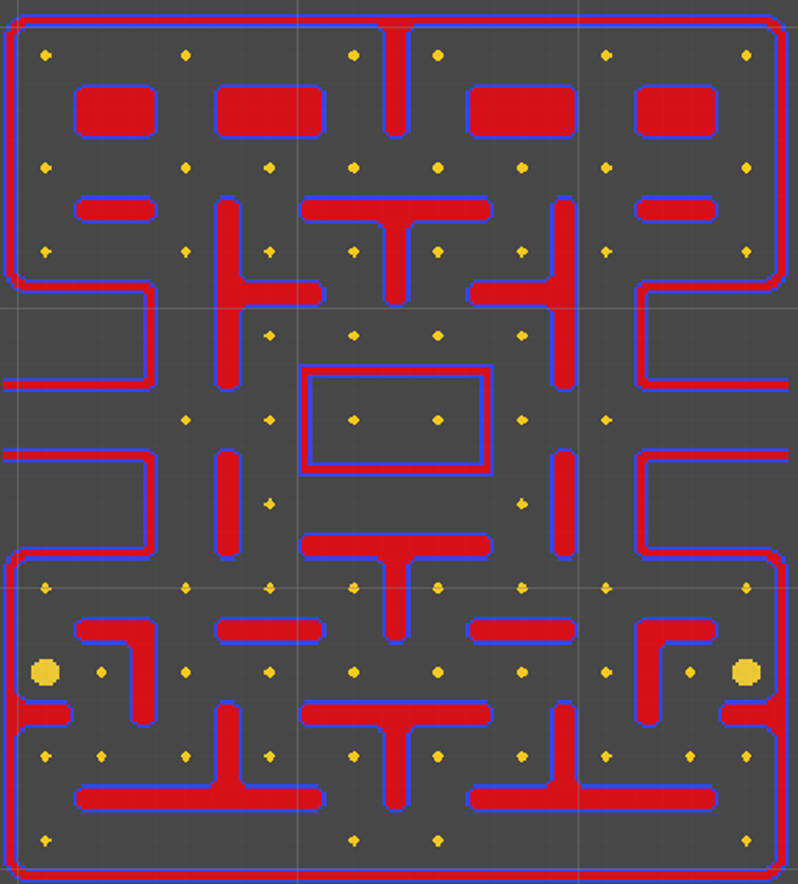
[38] Unity, 2021, Animator.runtimeAnimatorController Unity Documenation.

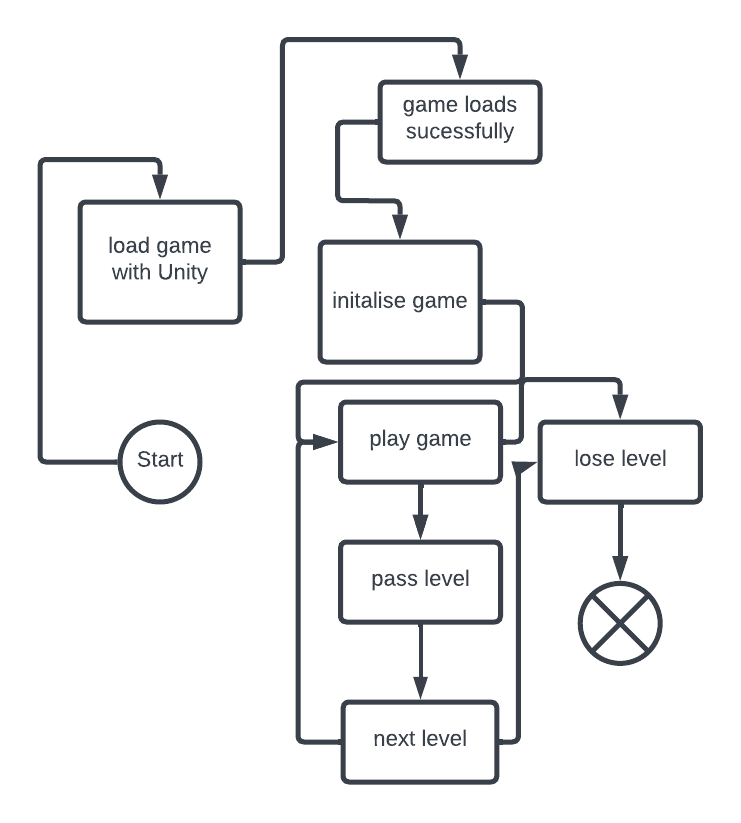
[50] PacMan Fandom (2022) Blinky,. Available at: <https://pacman.fandom.com/wiki/Blinky>

**Appendix** 1 This section shows the sprites of the ghosts

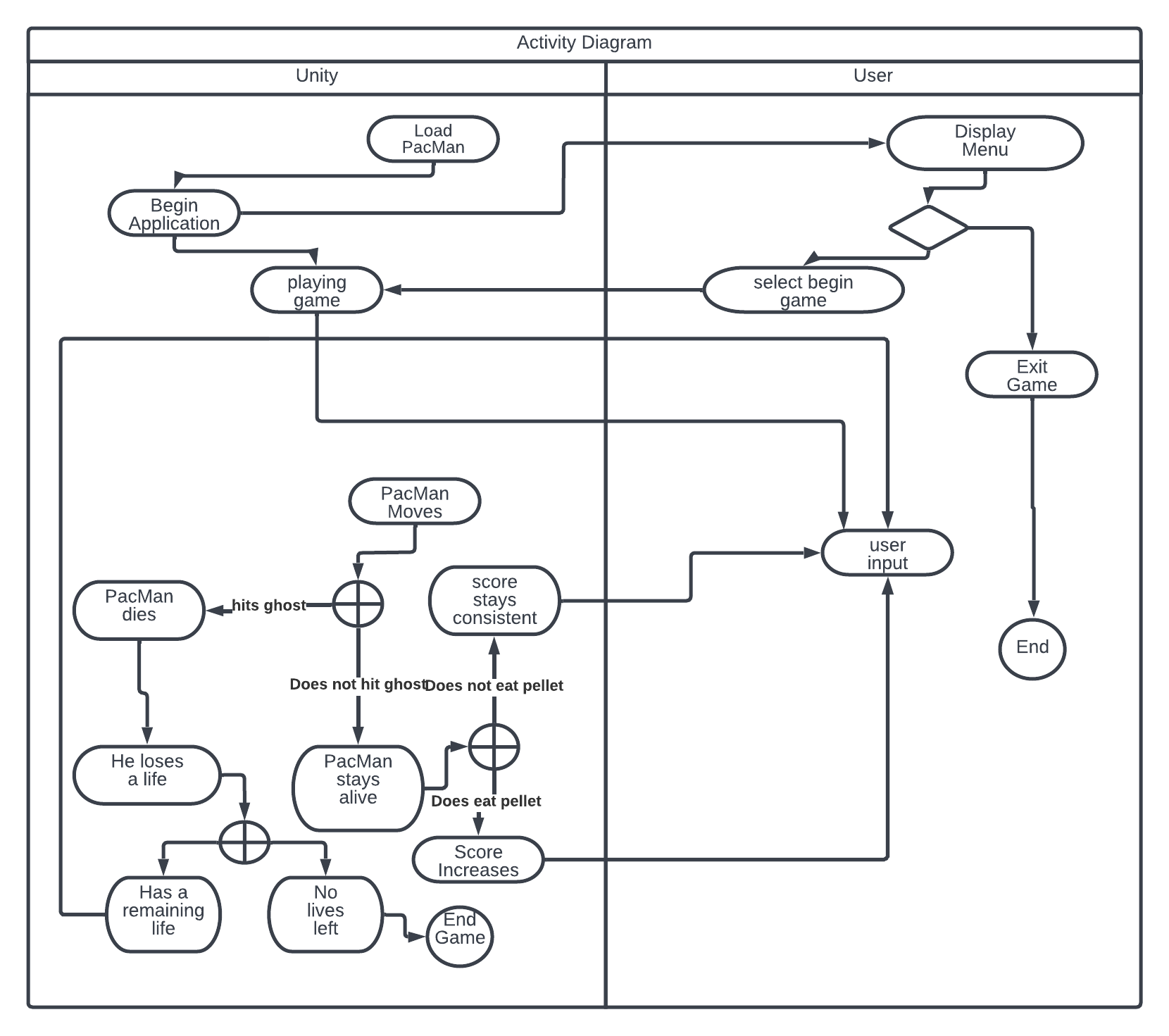


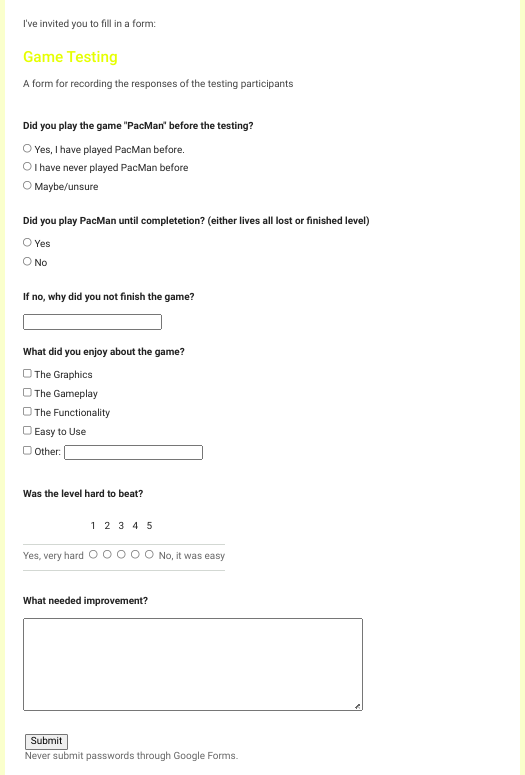
Appendix 2: This screenshot from the game is to show the waypoints at each point where the Player can change PacMan’s direction.



Appendix 3: This is a diagram to show the different states of the game.

Appendix four in an activity diagram to show the change between user actions and unity’s actions.



Appendix 5: A copy of the Survey used to collect responses from testing participants