PAC-WOMAN

Non-Examined-Assessment By:

Lauren Taylor

Candidate Number: 4664

CONTENTS

Executive summary:

In this project I have created a revamped version of Pac-man. I took the classic arcade game as an inspiration, and added some of my own functionality to create my own version of the game - PAC-WOMAN. I gathered inputs from girls to generate my objectives – with the aim of making the game more fun for them. The main improvements I added to the game were: I made it multiplayer with both players playing at the same time cooperatively and I included sophisticated randomly generated mazes for each level.
I have created this game using the game engine Unity as the basic framework and programmed the functionality for PAC-WOMAN in C# using a range of Object Oriented Programming techniques such as inheritance, and polymorphism as well as encapsulation. Testing has been performed against all off the identified objectives.

The most challenging part of creating this game was ensuring that every part of the randomly generated mazes could be reached by the players. I created my own tree traversal algorithm called the 'De-loop-inator' to do this. Other aspect that was particularly complex was the ghost chase algorithm – this needed to choose the ghosts' routes through the maze to try to catch the players (or run away from them).

The final result is a game that the target audience found fun to play.

1 ANALYSIS

1.1 Project statement

The classic arcade game 'PAC-man' is no longer being played widely nor is it a popular game anymore. For my project I intend to create a new, improved and revamped version of the original game and I shall call it 'PAC-woman'.

1.1.1 Observation of current game

Pac-man was originally an arcade game developed in the 1980's in Japan by Namco. The aim of the game is to go down every path of a maze and collect all the dots in order to progress to the next stage. However, whilst you try to collect all the dots (bits) you are chased by up to four ghosts. If they touch you then you lose a life. More specifically the key features of the game are:

- Maze:
  - no dead ends
  - made up from a collection of L shapes, T shapes and rectangles
  - Path has a width of 1 tile
  - There's a box that contains the ghosts in the centre – their home
- Only one pac-man played by the user. Originally controlled using a joy stick where the choice is to either move up, down, left or right.
- There are x4 ghosts that chase the pac-man. Each of them has their own 'personality' and tactics to attack the pac-man (they don't all just go for the pac-man all the time). The red ghost chases pac-man, the pink one aims for the point in front of pac-man, the blue sometimes heads for pac-man and other times just runs away and finally the orange one's movement is 'random'.
- If the player is hit by a ghost then they lose a life. Once all three lives are lost then the player dies and the game ends
- Once the player has eaten all of the bits on the maze (without losing all their lives) then they progress onto the next level – another maze. At each level the ghosts becomes more cunning and move faster making it harder to collect all of the bits without getting caught.
- Near the 4 corners of the maze there are Power Pellets that give the pac-man the power to eat the ghosts, for a short period of time that decreases. When it ends you return to normal play with the ghosts chasing pac-man again.
- The player gets points for every bit they eat. Additionally, when they eat ghosts they gain extra points)
- Pac-man can collect cherries which appear at random around the maze for a short time. If eaten then the player gains several more points than normal
- There is a max of 256 levels (limited by the hardware of the original arcade game with only 8 bits allocated to level counting)

1.2 Investigating the project's problem

Currently I have chosen young girls between the ages of 7-17 as my target audience/end user for this project. I have chosen them as my target audience because I want gaming, a 'boy' thing, to become considered more unisex. But also hopefully to inspire girls to get involved with computer science and show them that girls can develop games too! The two main girls I have selected to test my game out on are from two different age groups. I chose them because I wanted the opinion from both someone who already enjoys gaming but also someone whom doesn't really game at all. In doing so the purpose of the project is to bring together girls from different gaming backgrounds to enjoy playing this classic computer game.

1.2.1 Interviews

I interviewed 2 different girls (of different age groups) and asked them a few simple questions here were their responses:

Emma aged 10 & Thea aged 16

What's your background in gaming?

"I regularly play online multiplayer games (like Fortnite). It's a good fun way to hang out with friends or meet new people!"

"I'm not really into gaming. But I do enjoy a good puzzle or trying to solve Rubik cubes."

Have you played pac-man before?

"Yeah I've played the game a couple times before, I even had a go once at a retro arcade center that was fun"

"I have played the game before yes. But not many times, I got bored quickly"

What are your opinions on pac-man, how would you go about improving the game?

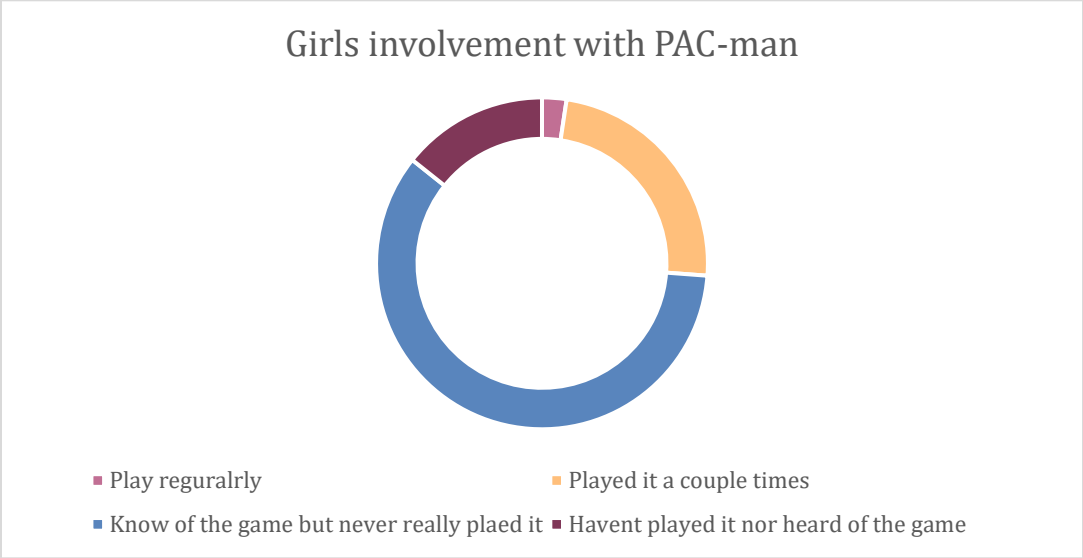
"My problem with pac-man is that you can't play it with multiple people. Gets really boring, lonely and repetitive after a while. I think that it would make the game much more enjoyable if there was an opportunity to play either with another person or against another person."

"My problem with pac-man is that its' mazes are too basic. I think that the game would improve greatly from having more complex and difficult mazes to that you

have to get through. Maybe include some actual dead ends where the player can get stuck in."

1.2.2 Questionnaires/surveys

I set up a short questionnaire where people could tick the statement that related to them the most. I then gathered 50 female students in my school to fill it out and these were the results that I received:



1.2.3 Research

From my research it doesn't seem that you can play pac-man in multiplayer mode unless it's on the Nintendo switch version of the game called 'PAC-MAN™ VS. (Free Multiplayer-only Ver.)'. Where you can 'become Pac-Man or a ghost and challenge up to three other players in classic gameplay with a multiplayer twist!'. However you can only play the game if you download it from the Nintendo eShop and you can only play it with other people who also already own a Nintendo switch.https://www.nintendo.co.uk/Games/Nintendo-Switch-download-software/PAC-MAN-VS-Free-Multiplayer-only-Ver--1248151.html

It's not entirely obvious whether or not the maze for pack man is randomly generated. I do know that the maze changes as one progressed through levels however I don't know how often or by how much seeing as I can barely get past the second round. Upon reading there does seem to be some kind of random generation of the mazes on pac-man but they all follow the same sort of constraints and never have any dead ends. Personally I think that it would make the game more exciting to try and include some dead ends.

During my research I also found this website which gives examples of tactics to win Pac-Man, found it interesting:
https://www.wikihow.com/Eat-Ghosts-in-Pacman-Without-Being-Caught

1.2.4 Summary of my findings

Based on the feedback from my interviews I identified that the there are two ways in which I should go about improving the game.

First, it needs to be more collaborative. The end users will enjoy the game more if it were multiplayer. It would be interesting to create a game where the two player had to work together to collect all the edibles and avoid being eaten by the ghosts. I anticipate that it will completely change the game dynamic and tactics.
The second way I will go about improving the game is to create new, randomly generated mazes for each new level of the game. This way the mazes shouldn't be predictable nor too easy to travel around. Adding a dynamic element to this aspect of the game will make the game less repetitive.

By addressing these two main improvements to the classical Pac-Man, will hopefully make the game more likely to be enjoyable for my target audience.

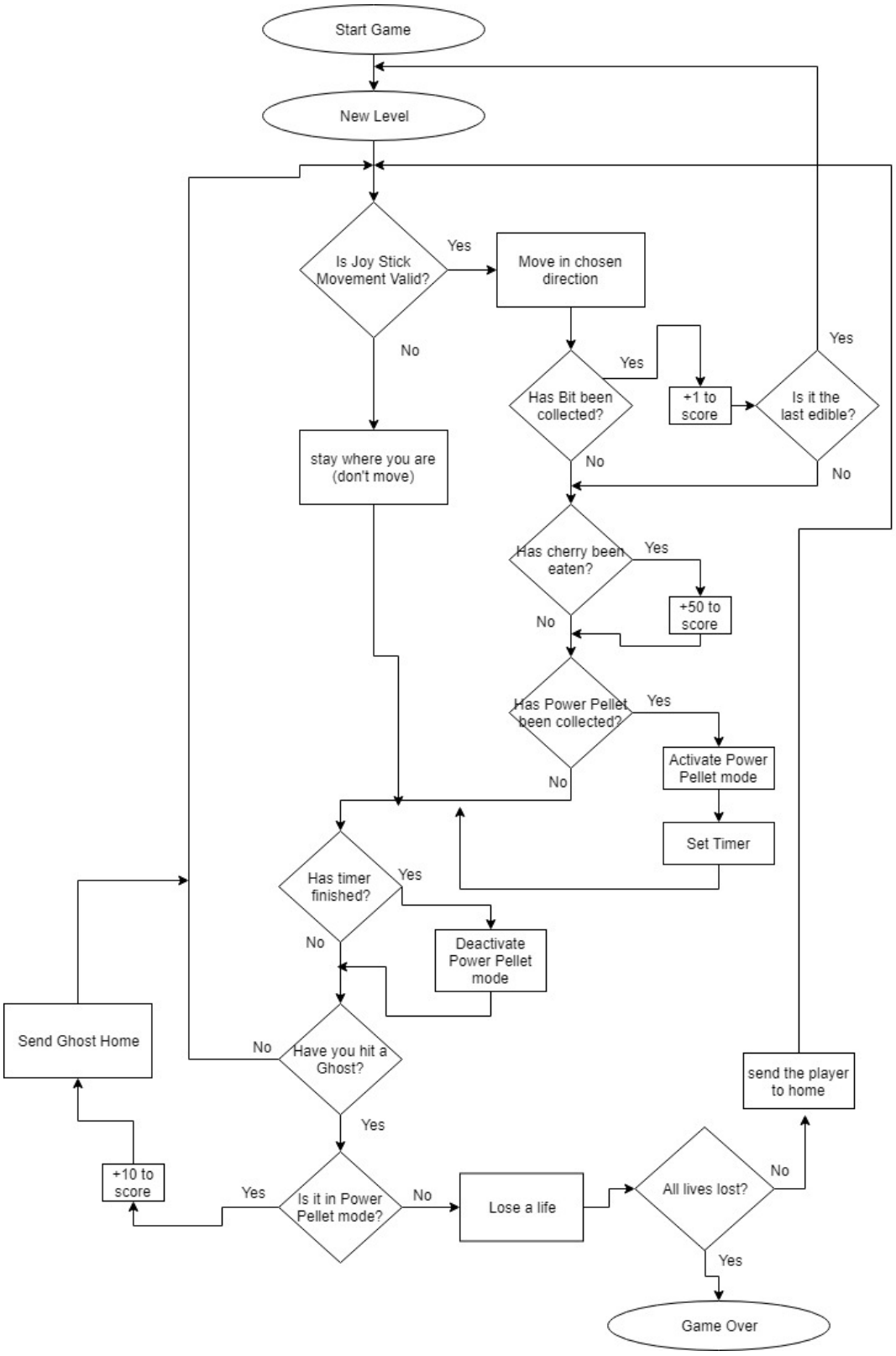
1.3 The problem to be solved

Pac-man was originally an arcade game that was first released in America in 1980 after being designed by Toru Iwatani in Japan. Today you can play pac-man in more ways and places than at an arcade. It has been developed as an app, pc, web game, Nintendo ds…. My project shall be to develop upon the variants of the classical game to create a new revamped pac-man game. Introducing some new features aimed at making it more accessible to girls. It is a classic game that has had huge appeal over many generations and so should be a good candidate for intriguing and entertaining girls. Based upon my research and feedback the two main aspects of the game that I would like to develop upon are:

- **Create a multiplayer mode:**
  Most games that are currently popular are multiplayer. This has become the case with the increase in the use of games as a method to hang out with friends (and make new ones) over the internet. Therefore the most enjoyable games to play with others allow multiple people to join in at once. The current Pacman is defiantly too much of a lonely game.
- **Generate more complex mazes randomly**
  In the current game, the mazes that pac-man runs around are disappointingly basic and repetitive. Likely because at the time the game was developed there wasn't much computing capability to generate more complex mazes for every individual level without the generation being prohibitively slow.

1.4 Modelling the existing system

1.4.1 System flowchart



1.4.2 Data dictionary

Data item	Data Type	Validation	Sample Data
Players name	string	<= 20 characters	Bob
Game level	integer		5
Time taken to finish game	time		4:22
Number of bits eaten	integer		276
Number of Power Pellets eaten	integer	<= 4	3
Number of ghosts eaten	integer		4
Number of cherries eaten	integer		2
Number of ghosts active in the maze	integer	<= 4	3
Total score @ end of game	integer		1098
Lives left	integer	<= 3	1
Joy stick movemenet	direction		Up

1.5 Objectives of the new system

Objective No.	Objective	Performance Criteria (how will you know if objective has been successful)
1	Allow the users to play in multiplayer mode with at one other person	Can two people play the game at the same time without disruption to the game? Need to have the same fun as a single player experience. Are there two pac-man players active at the same time?
2	The player controls the direction in which their pac-man travels (choosing between up down left or right)	Can the player move the pac-man based upon their keyboard inputs?
3	At each new level the game randomly generates a maze that allows the pac-man players and ghosts to travel around without	Do the mazes vary, and are they definitely random? Also ensure that the maze is actually playable, easy enough for the players to move

	getting too stuck. All parts of the maze should be accessible by the players	around in and that the players can reach all parts of the maze where there are edibles
4	The maze generation should increase in density/difficulty as you go up through the levels	Does the maze generation create more complex mazes as the players progress through the levels? Does it get harder to complete the level?
5	The score count displayed during the game is live and updates regularly	Does the score record accurately how many points have been gained?
6	The pac-man gains points as the pac-man collects: bits, ghosts and cherries	Does the score increase when items are collected?
7	Pac-man loses a life if eaten by one of the ghosts	Is a life lost when the pac-man touches a ghosts (when not in power pellet mode)
8	When a power pellet is eaten (for 10 seconds) the player is be able to eat the ghost, causing the ghost to respawn. and the player gains points foreating a ghost.	Can you eat a ghost (for 10 seconds) after eating a power pellet? Does the ghost respawn when eaten? Are points added to score after doing so?
9	When a power pellet is eaten the ghosts run away from the player	Do the ghosts move in a tactical format to avoid/ run away from the players when in power pellet mode?
10	The ghosts chase the pac-man based on the players specific position in normal mode (no power pellet)	Do the ghosts move in a tactical format to catch the player?
11	Certain ghosts should only get released based on which level the players are on. Each ghost should have slightly different personalities on how they chase the player	Do you get more ghosts as the levels progress? Do each of the ghosts have slightly different characteristics when chasing the players?
12	The level is complete when the player collects all of the edibles	Does the game progress to a new level once all edibles have been eaten (not including all cherries)?
13	Each player has their own set of lives.	Do the players lose lives independently? When one player has lost all their lives does the game continue with just the remaining player?
14	The game is over when both payers have lost all three lives	Does the game end when all the lives have been lost?
15	The game asks the players to enter a team name if their score qualifies as a high score	Does the game ask players to enter a team name only if their score is higher than the lowest high score on the current high score table?
16	Saves the top 10 highest scores	Does the game record high scores from previous games? Does this

		persist even when the game is shut down?
17	Players should be able to view the high score table even if they haven't achieved a high score	From the main menu can you view the current High score table?
18	Generates a league table that is regularly updated using a sorting algorithm	Is the high score ordered with the highest score at the top and lowest score at the bottom?

1.6 Possible alternative solutions to the problem

Instead of using the unity game engine I could have used OpenGL. Opnegl is an industry standard for high performance graphics with a range of possible implementations. I could have used it to present my game; it would have allowed me to do amazing graphics because OpenGL is mainly used for really high level rendering. However for this project I was not really interested in the graphics, it would have been over kill. If I used the OpenGL libraries I would still have had to program my own basic game engine for the scheduler and event handling. I didn't want my project to be focused upon trying to write my own game engine, it would take a lot of effort with not much to show for it. I was more interested in developing an actual game and was happy to use an existing game engine. I wanted to write a write a game rather than a game engine.

Using Unity almost felt like a no brainer. I was first introduced to it during my work experience at FireProof games (a commercial producer of games). I enjoyed using it there and found it easy enough to use - so stuck with it. Unity seemed the best platform because it integrated both graphics and scheduling, allowing me to focus on the achieving the objectives I had set for the game. The only slight dilemma is that I had to learn C#, where the only programming language I knew previously was python.

2 DESIGN

2.1 Critical path through the project

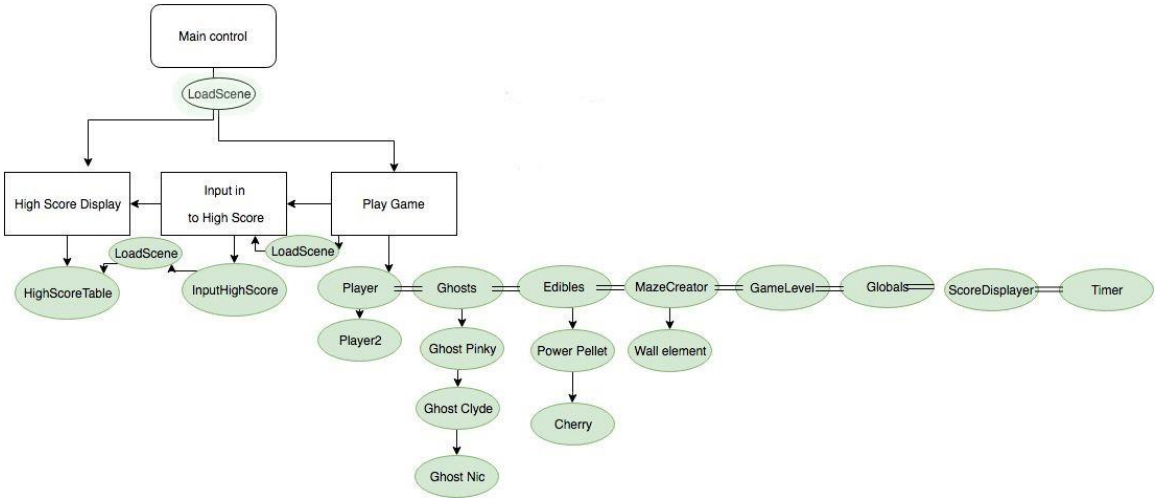
I intend on using an incremental method, so I will always have a game that you could use, but just kept adding in new functionality.

A plan of the order in which I intend to execute my project's objectives:

1. The player controls the direction in which pac-man travels using the up, left down, right keys.
2. The pac-man gains points as the pac-man collects: bits and cherry
3. The ghosts chase the pac-man based on the position of the players
4. Pac-man behaves differently when eats power pellet: can eat the ghosts
5. Pac-man loses a life if eaten by one of the ghosts
6. Create the multiplayer mode, so there are 2 players in the game at once working together to get through levels
7. Randomly generate mazes that allow the pac-man and ghosts to travel around without getting stuck
8. Decide on how will handle the level transitions; how will the game progress in difficulty as the players progress through the levels
9. Create the high score displayer and the opportunity to enter team name so their score can be added to the High Score table
10. Generate the algorithm to read from file, insert/sort the high score. Then overwrite back to the file. (Will require validation and exception handling)
11. Create the main menu options and connections between the different scenes

2.2 Overall system design

2.2.1 Top-down diagram



2.2.2 IPSO chart (input, process, storage, output)

Input	***Process***	***Storage***	***Output***
Choosing an option from the main menu (play game or view High Score table)	Which scene needs to be opened next	The chosen option	The scene which correspond to the chosen option
	Reading the High score table File. Extract the lowest high score for comparison	Store the values of the high score file into a list	Displays the high score list into a table format with columns, best score at the top
		The lowest of the high score	
Controlling the player using the arrow keys (and awsd)	Is movement valid (or will they collide with a wall)	The chosen direction of movement	Move the player sprite in chosen direction
	What is the current mode: normalPlay, powerPellet, endLevel or gameOver	The current mode	If mode is powerPellet: The ghost should flash colours
			If the mode is newLevel: displays the number of the level that was just completed
			If the mode is Gameover: Displays the text 'Game Over'
	Check if player has eaten : bit, cherry, power pellet or ghost	Increase the score based on the score value of the edible	Displays the current score and remove the edible that was eaten from the screen (or in the case of the ghost its randomly relocated)
	Check if player has been hit by a ghost (not in power pellet mode)	Reduce the life count of the player which was hit	Displays the current number of lives for each player
	Ghost chase algorithm	Chose direction of movement and the speed at which the ghost moves in that directions	Move the ghost in the calculated direction
	Check has power pellet timer run out	Store the timer and decrease it every second	Display the timer count down of how long the power pellet mode has left

Input	*Process*	*Storage*	*Output*
	Activate and randomly position the cherry every 10 secs	The new position of the cherry and whether it should be active or not	Display the cherry in chosen position
Achieving a new level	Check have all edibles been eaten	The current number of edibles	Display the number of edibles left
	Check the mode	The current mode should = newlevel	Display the new level number.
	Randomly generates a fresh maze (and increases the maze difficulty)	Store and instantiate the decided (de-looped) maze. The positions of the desired wall elements are stored into the array	Displays the fresh maze, using the arrays after the de-loop-inator has been applied to it
	Increase the number of ghosts	Which ghosts should be active	Display the currently active ghosts
	Re populate all the edibles	All the edibles needed	Display all the edibles
Ending the game	Check if the mode is Gameover	The mode	Displays the text 'Game Over'
	Check if score is eligible to be entered into the high score table	The lowest of the high scores from the file	Display the scene where can enter team name
	Insert and sort the score , with their team name, into the list	The team name and team score, rewrite the new version of the file which includes the latest high score values	Displays the team's name and score sorted into the high score table in the highScoreDisplay scene
Entering a Team Name (if eligible to enter High Score table)	Check if the length of the name is within the max length (20)	The inputted team name and their score	Display the team name as it's entered and when it has been sorted into the High score table

2.3 Algorithms

2.3.1 Ghost chase algorithm

The aim of the ghost chase algorithm is to catch the player (or run away from the player depending on the mode). This algorithm has a lot of scope to become quite complicated (see the discussion on alternatives below), however I chose to base it simply on the straight line distances between the player and the ghost.

Every frame the algorithm considers what the new distances would be if the ghost moved one step in each of the four directions (up down left and right). The algorithm then chooses to move the ghost in the direction that makes the linear distance between the either of the players the shortest (or the longest if the ghost is running away from the player; in power pellet mode). This algorithm is very effective at chasing the player when the game has little or no maze walls. However, when I began introducing the maze, the ghost would have difficulty getting out of dead ends, especially if the player was on the other side of the wall the ghost was banging its head against. What I did about this was allow the ghost to continue with the same 'best direction' over a couple of frames. In doing so allowing it to explore the same direction a little further in the hopes of it finding its way out of its rut. I also added a randomness of whether or not it would actually move in the direction that it had chosen as the optimal. This gave a nice visual effect as if the ghost was alive and exploring other possible directions but it also did help give it a chance to escape its dead ends. Once I had begun creating the levels and varying the maze densities, I realized that the ghosts were really effective in low density mazes. So I decreased their speed to be slower than the player in order to give the player a chance. However, when the mazes became more complex with higher densities of walls, the ghosts had a lot of difficulty finding and following the players, so I then increased their number and their speed. Making them quite deadly if they latched on to you! For a short while I had the idea of allowing the ghost to be able to pass through the walls of the maze with a low probability, but this made them too cunning and not as fun for the player.

There is potential to make the algorithm more elaborate by using the information about the maze to be used to determine the best direction to travel. However, this would require a lot of processing because the player is not still, the ghost would have to re-assess its path every frame. Also finding the shortest path between two points in a maze is quite difficult problem. It would also not be very easy to try and create an algorithm that would have to travel a long way in the wrong direction just to get out of a large dead end in order to then peruse the player. In order to address this processing load it may be better to work out which next 10 steps would bring the ghost closest to the player and use that..

This sort of approach may be worth investigating further in future. Nevertheless wouldn't want the ghost to be too good at hunting down the player, because the game

would then be over in no time. Much more fun to give the players a good chance. The current scheme works well, it is efficient and effective and leads to enjoyable game play.

View test videos 10 & 11 for a demonstration of the algorithm in action

2.3.1.1 Pseudocode of Ghost Chase Algorithm:

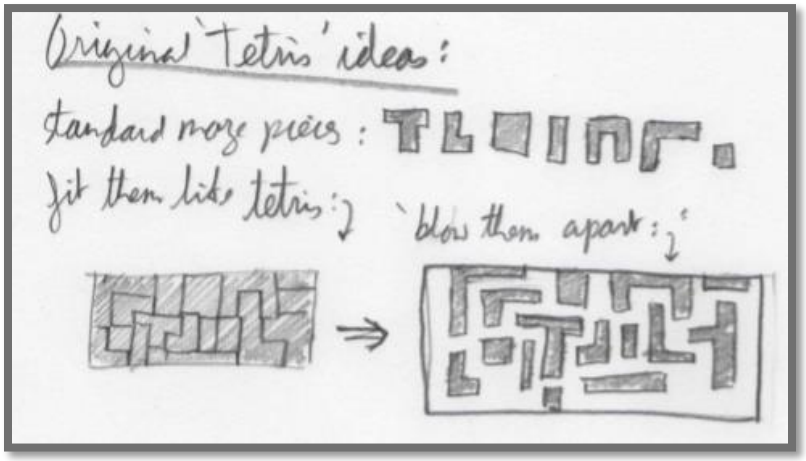
```
func      MoveGhost()
     {
          GameObject[] players;
          randomWait(10,25)
          {

               Vector2 targetPosition, trialPosition;
               distances = [];


               targetPosition = GetTargetPosition(players[i]);

               foreach (dir in Enum.Directions )
               {
                    trialPosition = GetDeltaPosition(dir, ghostPosition);
                    distances[dir] = DistanceBetween(trialPosition, targetPosition);
               }

               bestDirection = GetBestDirection(distances);
          }

          MoveInDirection(bestDirection);
     }
endFunc

func GetTargetPosition(GameObject target)
     {
          return target.transform.position;
     }
endFunc

func GetDeltaPosition(dir, CurrentPosition)
     {
          switch (dir)
          {
               case Directions is up:
                     CurrentPosition.y += step;

               case Directions is down:
                     CurrentPosition.y -= step;

               case Directions is left:
                     CurrentPosition.x -= step;

               case Directions is right:
                    CurrentPosition.x += step;

          }

          return CurrentPosition;
     }
endFunc

func DistanceBetween(trialPosition, targetPosition)
    {
        return.Sqrt((trialPosition.x - targetPosition.x)^2 + (trialPosition.y -
targetPosition.y)^2);

    }
endFunc
```

```
func Directions GetBestDirection(distances[])
    {


        if (Mode == normalPlay)
        {

            for (i  in range 0 to NumberOfPlayerAlive)
            {
                foreach (dir in Enum.Directions )
                {
                    if (distances[dir] < distances[(best])
                    {
               // only randomly updates the best direction
                 // i.e. dosen't always move in the best direction
               //giving it an opportunity to escape dead ends that its caught up in
                        if (randomNumber(1, 100) < 65)
                        {
                            best = dir;

                        }
                    }

                }
            }


        }

         if (Mode == powerPellet)
        {

            for (i in range 0 to NumberOfPlayerAlive)
            {
                foreach (dir in Enum.Directions )
                {
                    if (distances[dir] < distances[(best])
                    {
                        if (randomNumber(1, 100) < 65)
                        {
                            best = dir;

                        }

                    }

                }
            }
        }

        return best;
    }
endFunc

func MoveInDirection(Directions bestDirection)
    {

         switch (bestDirection)
        {
            case Directions is up:
                velocity = (0, ghostSpeed);

            case Directions is down:
                velocity = (0, -ghostSpeed);

            case Directions is left:
                velocity =(-ghostSpeed, 0);

            case Directions is right:
                velocity = (ghostSpeed, 0);

        }
    }
```

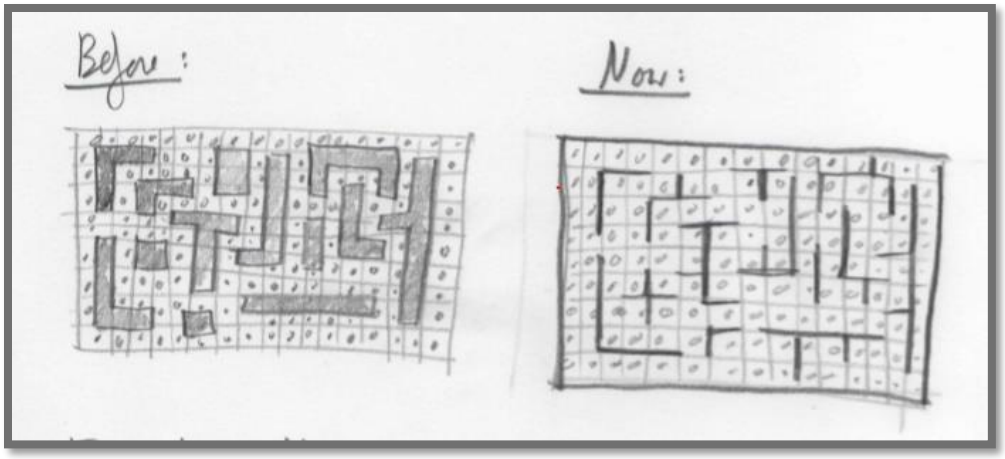
```
endFunc
```

2.3.2 Maze Algorithm design

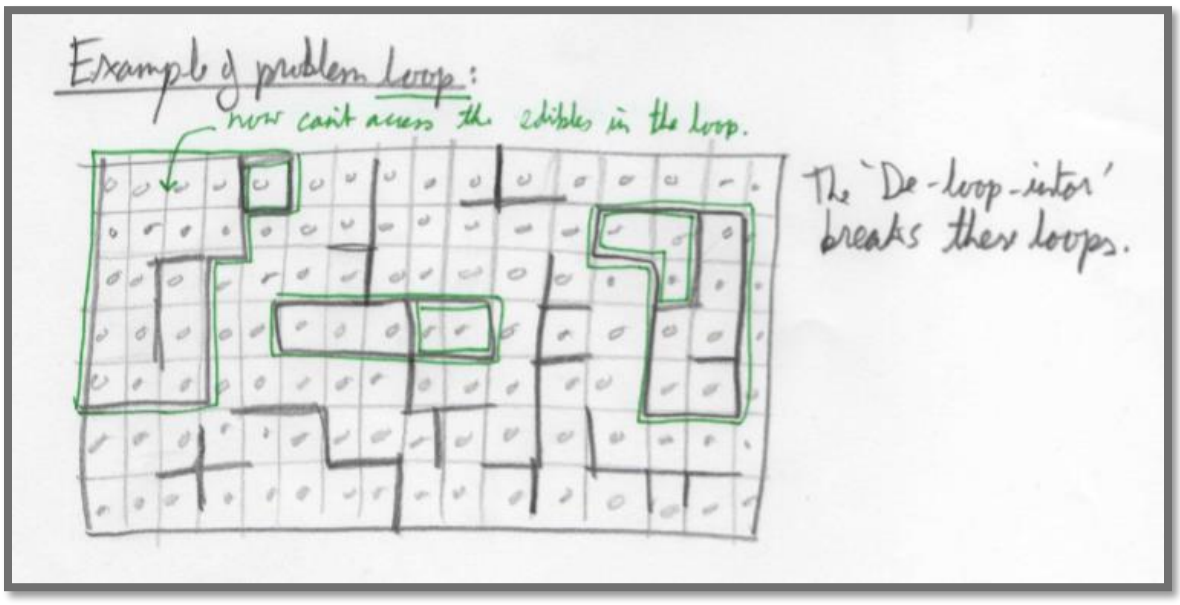
Originally, the idea for my randomly generated maze was to create a collection of standard maze pieces then create an algorithm that was similar to Tetris to put the pieces together. Once it had then filled out a certain space with pieces, it would 'blow' them apart to create spaces between them that would be the new maze. But upon trying to create an algorithm to do this I realized that this could be a complex process to implement, therefore I abandoned that ideas and began brain storming again to come up with a new method.



When thinking about generating the maze I was also considering how I would populate the maze with all the edibles. Amid those thoughts I decided it would be sensible and easiest to make the maze walls lie on the grid lines and have the edibles populate the empty spaces in-between (inside the grid's square spaces). Originally, the maze pieces were going to be solid shapes with depth that would occupy multiple squares on the grid, with the remaining space left have the paths. However, I decided against the walls with depth because it made it a lot trickier to make the maze big enough for the player to pass around and still be a complex and interesting enough.



Once I had decided that I was now going to use wall elements that lay in-between, like lines instead of wall pieces, I thought of how I could randomly generate it. What I came up with was simple, I would randomly populate the grid with horizontal and vertical wall elements. This also had the advantage of allowing the same fixed full grid of edibles to be used for every level. Most of the time this created a decent maze, although I did notice a common problem. If the wall elements managed to create a closed shape within the maze that would create a section of the maze that the player would not be able to access (or even trap the player into a shape that they wouldn't be able to escape). I was no longer phased by the idea of including dead ends into the maze (in the original Pac-man game there are no dead ends in the mazes) because it would make it a trickier route to transverse for the player. So I had to come up with an algorithm for breaking the closed loops in the maze (a De-loop-inator).



Instead of processing the gaps between the walls in order to create the maze. I would process the actual wall elements in the maze. I identified that the walls of the maze were broken up into a collection of small trees.

Even though I had been recently studying tree traversal algorithms (both in computer science and in Decision maths) they had all been using noncyclic graphs. My problem was to try and convert a cyclic graph into a noncyclic version, a de-looped version. So I came up with my own tree traversal and correction algorithm for creating noncyclical trees out of cyclical ones. The approach is as follows:

Each tree from the collection of 'trees' that made up the maze could be individually traversed to check for loops. When on a tree, traverse each wall element between the connected nodes. At a node the algorithm would, first record it as 'visited'. Then choose the next clockwise direction (up, right, down, left) with an existing wall element. Once it had chosen a wall element to traverse it would perform some checks before traversing to the next node:

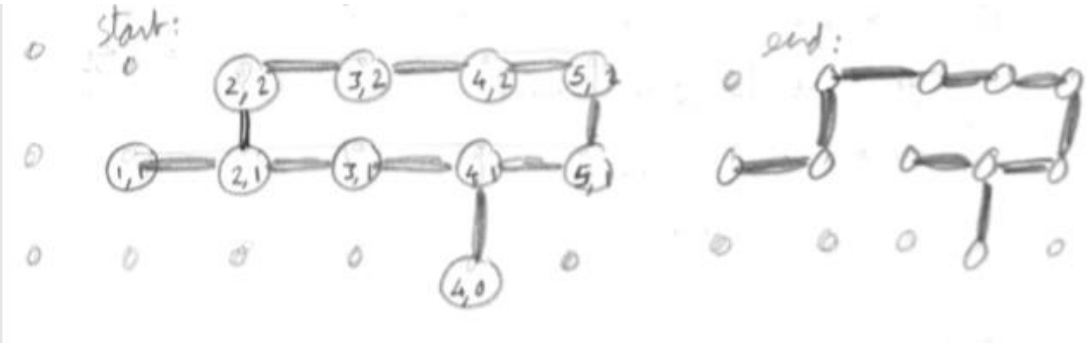
- Has the node, that would be reached if the wall was traversed, been visited before?
  - If NO (the next node has not been visited before): the chosen direction is okay to traverse. So move to that new node
  - If YES (the next node has been visited before):
    CHECK: has the wall element you're about to traverse been traversed before? (probably from the other direction)
    - If YES (wall element has been traversed before): this has not created a loop you are just retracing your step and returning to node that has already been visited, but needs to be visited again in order to get to all the other remaining nodes.
    - If NO (the wall element has not been traversed before): this HAS created a loop and therefore needs to be broken. So instead of traversing the proposed wall element delete it, in doing so breaking the loop.

Keep going on picking the next clockwise direction and repeat these checks at every node to be visited until return to the start node. When you successfully return to the node you started at you know that the whole of the tree has been explored if the visit count at the start node is equal to the number of wall elements connected to it.

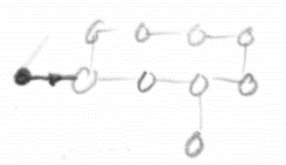
Once it has de-looped a tree, it would then find a new starting node to a new tree in the maze. Any node on a new tree will do as a starting node so just search for a node that has not been visited before. Once a new start node is found perform the de-loop traversal upon the new tree. The whole maze would be de-looped

once all the trees that it was made up from had been de-looped. You could check for remaining trees to be explored by my making sure that all 'alive' nodes (nodes that have wall elements attached to them) have been visited.

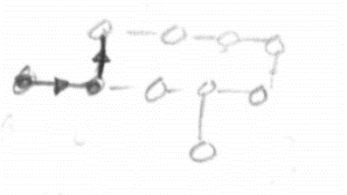
2.3.2.1 De-Loop-Inator Diagram example:



If (1,1) node is chosen as the *start node*.

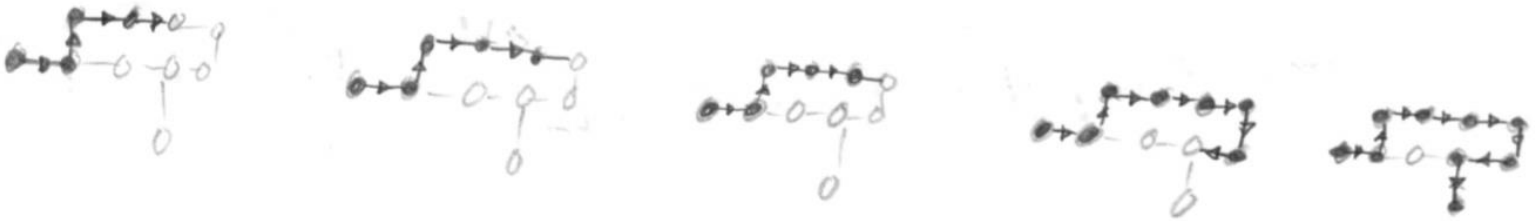


1. The first possible available path is in the direction Right & (2,1) has never been visited, so traverse the path from (1,1) to (2,1).



2. As Left was the direction came in from, the next clockwise direction is Up, to (2,2).
   (2,2) has never been visited before, so travers the path towards it.

3. The same logic from the previous steps ^^ are repeated till reach (4,0)



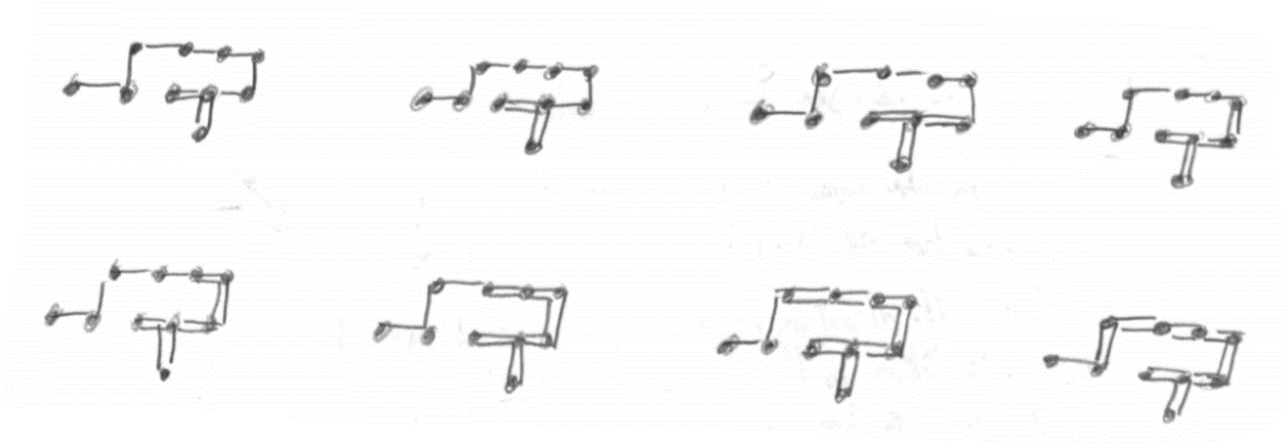
4. When reaches (4,0) the only available path is Up, back to (4,1) which has already been visited! However, the algorithm knows that it's NOT a loop because even though the node has been visited before the path has also been visited before, just in the opposite direction.

5. Then continue from (4,1) to (3,1) as normal

6. However when reaches (3,1) the next clockwise path to take is Left to (2,1), which has already been visited BUT the path between (2,1) & (3,1) has NOT been traversed before. Which means that in order to have reached this node again would have created a loop! (we don't want any loops)

7. Instead of traversing the path between (2,1) & (3,1). Delete the path and remove it form the maze.

8. Once the path has been deleted, recalculate the next clockwise path and backtrack until reach the start node again.

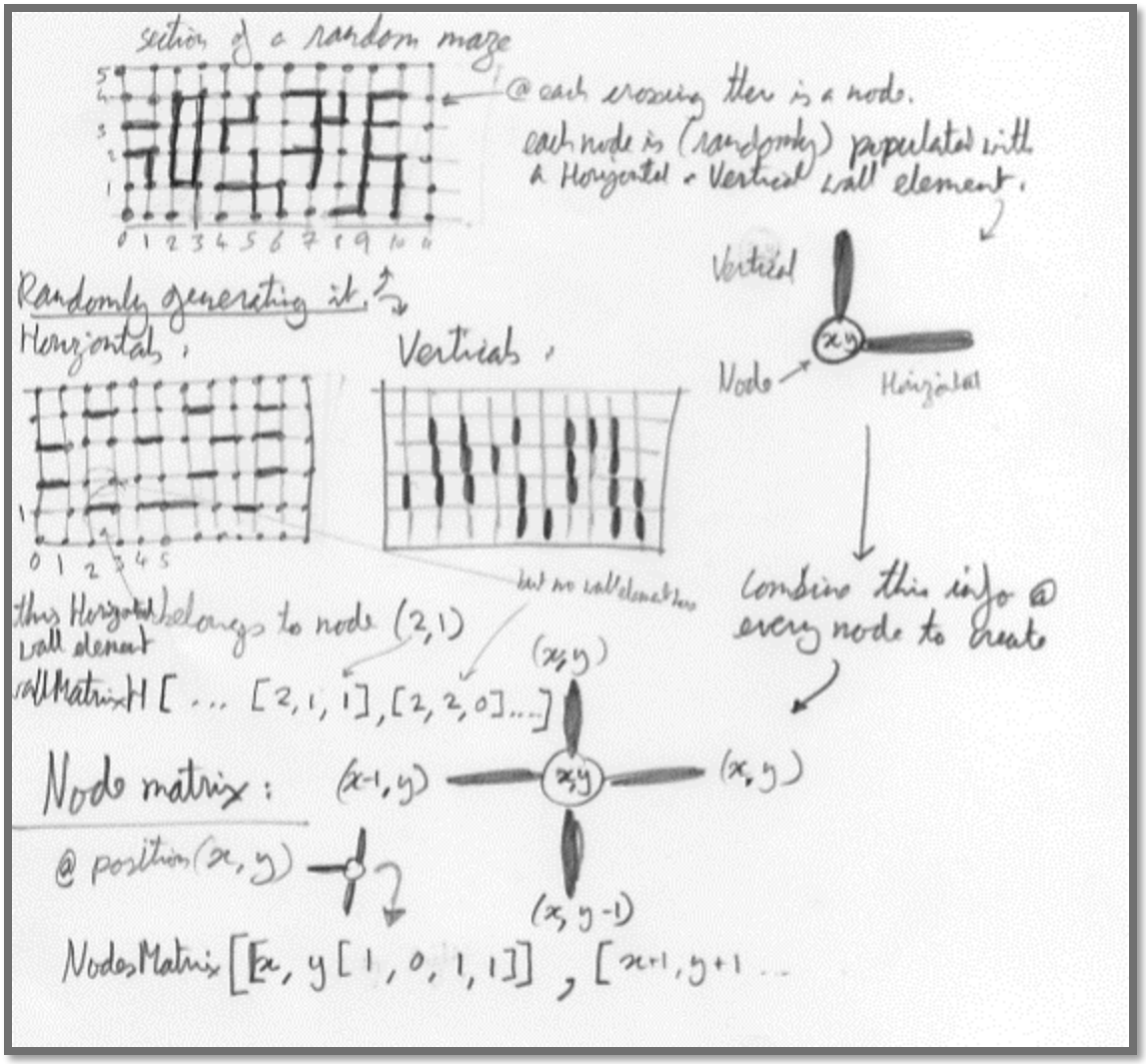


9. Know that the whole tree has been traversed when all the paths at the start node have been full (in both directions) traversed.

In order to implement this algorithm I needed a way of recording all the data about the randomly generated maze and storing it so that it could be traversed and corrected. As the maze is based on a grid of squares I settled on using a set of multidimensional arrays (like matrices) that I would use to represent the grid populated with the wall elements. When randomly generating the maze the first arrays to be populated were `wallsMatrixH` and `wallsMatrixV.` They were indexed by their node position and would store a 1 if the (Vertical or Horizontal) wall element was populated at that node. And a 0 if the wall element was not populated at that node.

Once the wallMatrixs were both populated they are combined to create the nodes matrix. The nodeMatrix stores for each node whether or not there is a wall element up, right, down, left at that node. (This does mean that the wall elements are doubled up because, for example, an Up element for one node is the same wall as the Down wall element for the node above it. Both nodes record a wall element present and they're both correspond to the same wall element in the maze.). This later comes in handy because when 'de-loop-ifying' the maze you need to know if the wall element has been traversed in both directions or not.

The two other Matrices that I created were: `nodeVisitCountMatrix and pathTraversalCountMatrix.` These two matrices are used in the de-loop-inator algorithm. The `nodeVisitCountMatrix` is indexed by the node's position it then stores how many times that particular node has been visited. `pathTraversalCountMatrix` keeps track of how many times each wall element attached to a specific node has been traversed. At the indexed positions it has 4 elements for the four corresponding directions for the possible wall elements. So each wall element will have a count of how many times it has been traversed. This is used in the de-loop-inator to determine whether traversing the wall element would have created a loop in the tree.



2.3.2.2 Pseudocode of Maze Algorithm

High level summary of the algorithm:

```
//--------MAZE CORRECTION ALGORITHM: ----TO DE-LOOP-IFY------------------
DE-LOOP-INATOR------------------
    //startNode = StartNodeFinder( NodesMatrix)
    //DirectionOfTraversal = ChooseTraversal(startNode,left) // checks for:
existence of wall elements and whether their node has been visited, chooses the
next one going clockwise
    //loops until completeTraversal
    //  DirectionCameFrom = drictionConversion(DirectionOfTraversal)
    //  newNode = TraverseToNode(DirectionOfTraversal, NodeMatrix)
    //  DirectionOfTraversal = ChooseTraversal(newNode, DirectionCameFrom)

    //ChooseTraversal(currentNode, DirectionCameFrom):
    //  whichWallDoesItHave = []
    //  tempNewNode = NextClockwise(whichWallDoesItHave)
    //  Visited? = CheckifVisted(tempNewNode)
    //  IF Visited? == Flase: //or maybe = 0 (see notes)
    //      return tempNewNode
    //  ELSE:
```

```
        //      IF checkPath(tempNewNode) == goingBack: // OR already has a
    'forward' to that node
        //          return tempNewNode
        //      ELIF checkPath(tempNewNode) == closedLoop: // OR there is NO
    'forward' to that node
        //          wantedWallElement = FindWhichWallElementToDelete(tempNewNode,
    currentNode )
        //          DeleteWallElement(tempNewNode, wantedWallElement) //deletes the
    connecting wall element that closes the loop. dont forget needs to be removed
        //          return currentNode
```

Detailed version

```
   wallsMatrixH = new int[Constants.mazeWidth, Constants.mazeHeight]; // 15 x 8
(x by y) matrix for the horizintal wall elemnts 0 no wall & 1 is there is a
wall
   wallsMatrixV = new int[Constants.mazeWidth, Constants.mazeHeight]; // ditto
for vertical walls

    /*nodesMatrix:
     * 16x9 because need an n+1 walls to bound n nodes (need an extra column on
the right and an extra row at the bottom)
     * n-n-n requires 3 nodes but 2 walls
     * the z dimension is for the Up = 0 Right = 1 Down = 2 left = 3 walls at
each node  (i.e. increasing clocksiwse will work with modulo 3 to loop round
Directions)
    */
    nodesMatrix = new int[Constants.mazeWidth+1, Constants.mazeHeight+1, 4];

    // variables to manage the corrections of closed loops in the maze
    nodeVisitCountMatrix = new int[Constants.mazeWidth + 1,
Constants.mazeHeight + 1]; // like node mtarix but holding the number of times
each node has been visited
    pathTraversalCountMatrix = new int[Constants.mazeWidth + 1,
Constants.mazeHeight + 1, 4]; // like the node matrix but holding the count of
the number of times that each path for a node has been traversed


for (j in range Constants.yBottom to Constants.yTop)
{
    for (i range Constants.xLeft to Constants.xRight)
    {

        //populates the Wall matrices randomly with 1s
        if (randomNumber(1, 100) < wallProbability)
        {
            //have the 'i + shift constants' in order to shift the postioins
into matrix indexs; to name the nodes
            wallsMatrixH[i + Constants.xShift, j + Constants.yShift] = 1;
        }

        if (randomNumber(1, 100) < wallProbability )
        {
            wallsMatrixV[i + Constants.xShift, j + Constants.yShift] = 1;
        }

    }
}

NodeMatrixCreator();

DeLoopInator();
```

```
func DeLoopInator()
    {

        //will de-loop-ify for each tree in the maze, when cant find any new
trees it will be finished

        //loop over the trees in a maze
         //when true means it has found a new tree i.e. a node that has not been
visited before
        while (StartNodeFinder(out iStartNode, out jStartNode) == true) //out =
the out paramter e.i. addintional return values
        {
             // increments the visit count by 1 for the start node of the tree
            ++ nodeVisitCountMatrix[iStartNode, jStartNode];

            iOfNode = iStartNode; //initiliaizing the values for the tree
traversal loop
            jOfNode = jStartNode;
            directionOfTraversal = 3;//using 3 because want it to start the
traversal coming in from the left

            //loops over the nodes in a tree, until completeTraversal of the
tree
            //ChooseTraversal: picks a new direction to traverse, returns flase
when the whole tree has been traversed
            //NOTE: as well as chousing the node to traverse to,
ChooseTraversal also breaks any closed maze loops
            while (ChooseTraversal(iOfNode, jOfNode, directionOfTraversal, out
iOfNewNode, out jOfNewNode, out directionCameFromNew) == true)
            {

                //once has chosen the new node to traverse to, it moves to it
i.e. sets it as the current node
                iOfNode = iOfNewNode;
                jOfNode = jOfNewNode;
                directionOfTraversal = directionCameFromNew;
            }

        }

    }
endFunc


//IsNodeAlive: not all nodes have wall elements attached to them, this
determines wether or not it has any wall elements
func    bool IsNodeAlive(i, j)
    {
        numberOfWallElements = 0;
        for (k in range 0 to 3)
        {
            numberOfWallElements += nodesMatrix[i, j, k];
        }
        if (numberOfWallElements > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

```
endFunc

 // StartNodeFinder: finding a node to start on i.e. for a tree that hasnt yet
been de-loop-ified
funcStartNodeFinder(out iStartNode, out jStartNode)
    {

        for (j in range 0 to Constants.mazeHeight + 1)
        {
            for (i in range 0 to Constants.mazeWidth + 1)
            {
                // checks that the node has walls connected to it AND that has
never been visited i.e. start of a tree
                if (IsNodeAlive(i, j) == true && nodeVisitCountMatrix[i, j] ==
0)
                {
                    iStartNode = i;
                    jStartNode = j;
                    return true; // has found a start node
                }

            }


        }
        return false;// has not found a start node, the whole maze has been
deloopified
    }
endFunc


    /* ChooseTraversal:
     * chooses a candidate direction
     * finds the corresponding node to the direction
     * checks if its has ever been visited before:
     *      never been visted before=okay to traverse to it
     *
     *
     */

funcChooseTraversal (iOfNode, jOfNode, directionCameFrom, out iOfNewNode, out
jOfNewNode, out directionCameFromNew)
    {
        tempDirectionNewpath, numberOfWallElementsAtNode=0;

        tempDirectionNewpath = NextClockwise(iOfNode, jOfNode,
directionCameFrom);//candidate direction
        directionCameFromNew = (tempDirectionNewpath + 2) % 4;//based on the
candidate direction it works out what its direction came from would be if it it
moved to this node

        UsePathToFindNode(iOfNode, jOfNode, tempDirectionNewpath, out
iOfNewNode, out jOfNewNode);//returns candidate node

        //the candidate node have never been visited
        if (CheckifVistedNode(iOfNewNode, jOfNewNode) == false)
        {

            ++nodeVisitCountMatrix[iOfNewNode, jOfNewNode];
            ++pathTraversalCountMatrix[iOfNode, jOfNode, tempDirectionNewpath];

            return true;
        }
```

```
        else //candidate node HAS been visited before
        {
            //      check if tree traversal is complete
            bool alldone = true;
            for (k in range 0 to 3)
            {
                if (pathTraversalCountMatrix[iOfNewNode, jOfNewNode, k] == 0 &&
nodesMatrix[iOfNewNode, jOfNewNode, k] > 0)
                 {
                     alldone = false;
                 }
            }
            if (alldone)
            {
                return false;
            }

            //calculates the number of wall elemenets at the node
            for (k in range 0 to 3) { numberOfWallElementsAtNode +=
nodesMatrix[iOfNewNode, jOfNewNode, k]; }
            //if back at strat node and alls its wall elements have been
visited then
            if (iOfNewNode == iStartNode && jOfNewNode == jStartNode &&
nodeVisitCountMatrix[iOfNewNode, jOfNewNode] == numberOfWallElementsAtNode)
            {
                // finished the tree
                return false;
            }


            //checks if the path its about to traverse has already been
traversed, but in the opposite direction
            //this case is NOT A LOOP
            if (CheckifVistedNode(iOfNewNode, jOfNewNode) == true &&
pathTraversalCountMatrix[iOfNewNode, jOfNewNode, (tempDirectionNewpath + 2) %
4] == 1)
            {
                ++nodeVisitCountMatrix[iOfNewNode, jOfNewNode];
                ++pathTraversalCountMatrix[iOfNode, jOfNode,
tempDirectionNewpath];
                return true;
            }

            // the node has been visited and the path has never been traversed
            //this case IS A LOOP, need deloopifying
            else if (CheckifVistedNode(iOfNewNode, jOfNewNode) == true &&
pathTraversalCountMatrix[iOfNewNode, jOfNewNode, (tempDirectionNewpath + 2) %
4] == 0)
            {
                //deletes the connecting wall element that closes the loop
                DeleteWallElement(iOfNode, jOfNode, tempDirectionNewpath,
iOfNewNode,jOfNewNode);

                iOfNewNode = iOfNode;
                jOfNewNode = jOfNode;
                directionCameFromNew = directionCameFrom;
                return true;
            }

        }

        return false;

    }
endFunc
```

```
 //calculates the next clockwise direction from the one you came in on
funcNextClockwise(iOfNode, jOfNode, directionCameFrom)
    {
        for (a in range 1 to 4)
        {

            directionGoingTo = (directionCameFrom + a) % 4;
            if (nodesMatrix[iOfNode, jOfNode, directionGoingTo] == 1)
            {
                return directionGoingTo;
            }
        }
    }
endFunc


funcUsePathToFindNode(iOfNode, jOfNode, direction, out iOfNewNode, out
jOfNewNode)
    {
        int i = -1;
        int j = -1;
        if(direction == 0) //up
        {
            i = iOfNode;
            j = jOfNode + 1;
        }

        else if (direction == 1)//right
        {
            i = iOfNode + 1;
            j = jOfNode;
        }

        else if (direction == 2)//down
        {
            i = iOfNode;
            j= jOfNode - 1;
        }

        else if (direction == 3)//left
        {
            i = iOfNode - 1;
            j = jOfNode;
        }
        iOfNewNode = i;
        jOfNewNode = j;
    }
endFunc


funcCheckifVistedNode(iOfNode, jOfNode) //returns false ONLY if has never been
visited
    {
       if( nodeVisitCountMatrix[iOfNode, jOfNode] == 0)
        {
            return false;
        }
        return true;
    }
endFunc

// DeleteWallElement; removes the wall element from all the matrices
func DeleteWallElement(iOfNode, jOfNode, DirectionPath, iOfNewNode, jOfNewNode)
    {
```

```
        pathTraversalCountMatrix[iOfNode, jOfNode, DirectionPath] = 0;
        pathTraversalCountMatrix[iOfNewNode, jOfNewNode, (DirectionPath + 2) %
4] = 0;

        nodesMatrix[iOfNode, jOfNode, DirectionPath] = 0;
        nodesMatrix[iOfNewNode, jOfNewNode, (DirectionPath + 2) % 4] = 0;

        if (DirectionPath == 0)
        {
            wallsMatrixV[iOfNode, jOfNode] = 0;//removing the Up wall element
from the node
        }
        if (DirectionPath == 1)
        {
            wallsMatrixH[iOfNode, jOfNode] = 0;//removing the Right wall
element from the node
        }
        if (DirectionPath == 2)
        {
            wallsMatrixV[iOfNode, jOfNode-1] = 0;//removing the Down wall
element from the node
        }
        if (DirectionPath == 3)
        {
            wallsMatrixH[iOfNode-1, jOfNode] = 0;//removing the Left wall
element from the node
        }

    }
endFunc
```

2.4 Data structures

The majority of the data used in the program can be stored in simple data types, integers, floats and enumerated types.

However the Maze generation Class uses requires a number of cyclic tree structures in order to allow the maze to be defined and traversed in order to identify any closed loops and allow them to be opened creating an acyclic tree.

As the maze is to be displayed in a regular square grid, it is convenient to use multidimensional arrays to store the necessary maze structural and traversal information.

The matrices used are listed below with annotations. These can be related directly to the maze traversal and de-loop-ify algorithms described in the section above.

```
    private int[,] wallsMatrixH = new int[15, 8]; // 15 x 8 (x by y) matrix for
    the horizintal wall elemnts 0 no wall & 1 is there is a wall
    private int[,] wallsMatrixV = new int[15, 8]; // ditto for vertical walls

    /*nodesMatrix:
     * 16x9 because need an n+1 walls to bound n nodes (need an extra column on
the right and an extra row at the bottom)
     * n-n-n requires 3 nodes but 2 walls
     * the z dimension is for the Up = 0 Right = 1 Down = 2 left = 3 walls at each
node  (i.e. increasing clocksiwse will work with modulo 3 to loop round
Directions)
    */
    private int[,,] nodesMatrix = new int[16, 9, 4];

    // variables to manage the corrections of closed loops in the maze
    private int[,] nodeVisitCountMatrix = new int[16, 9]; // like node mtarix but
holding the number of times each node has been visited
    private int[,,] pathTraversalCountMatrix = new int[16, 9, 4]; // like the node
matrix but holding the count of the number of times that each path for a node has
been traversed
```

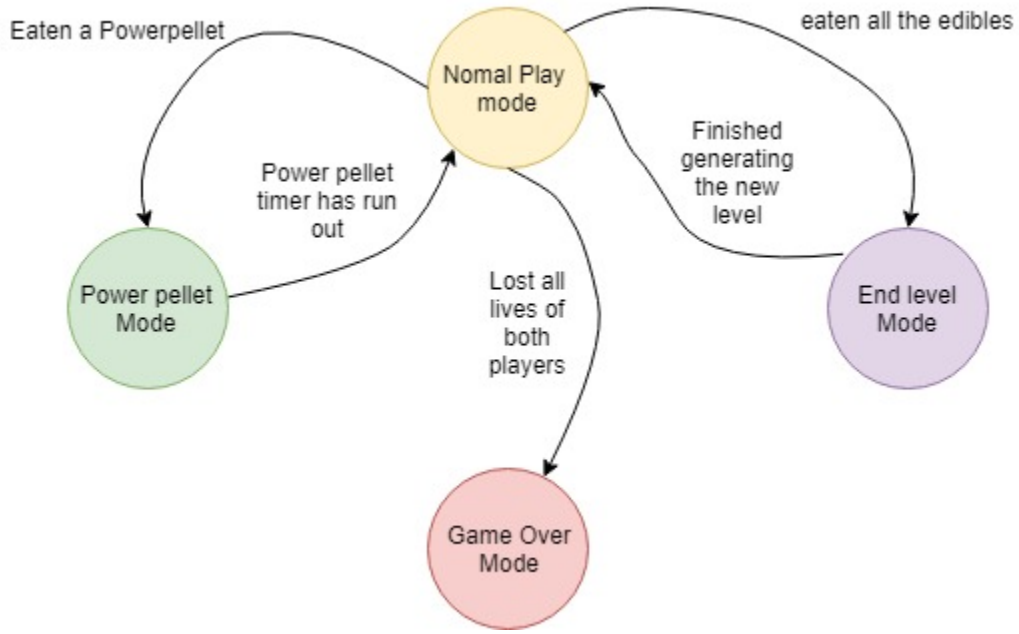
Also used a couple of Enumerated types which I declared for descriptive convenience.

These are also listed below:

```
private enum Directions {up, down, left, right};
```

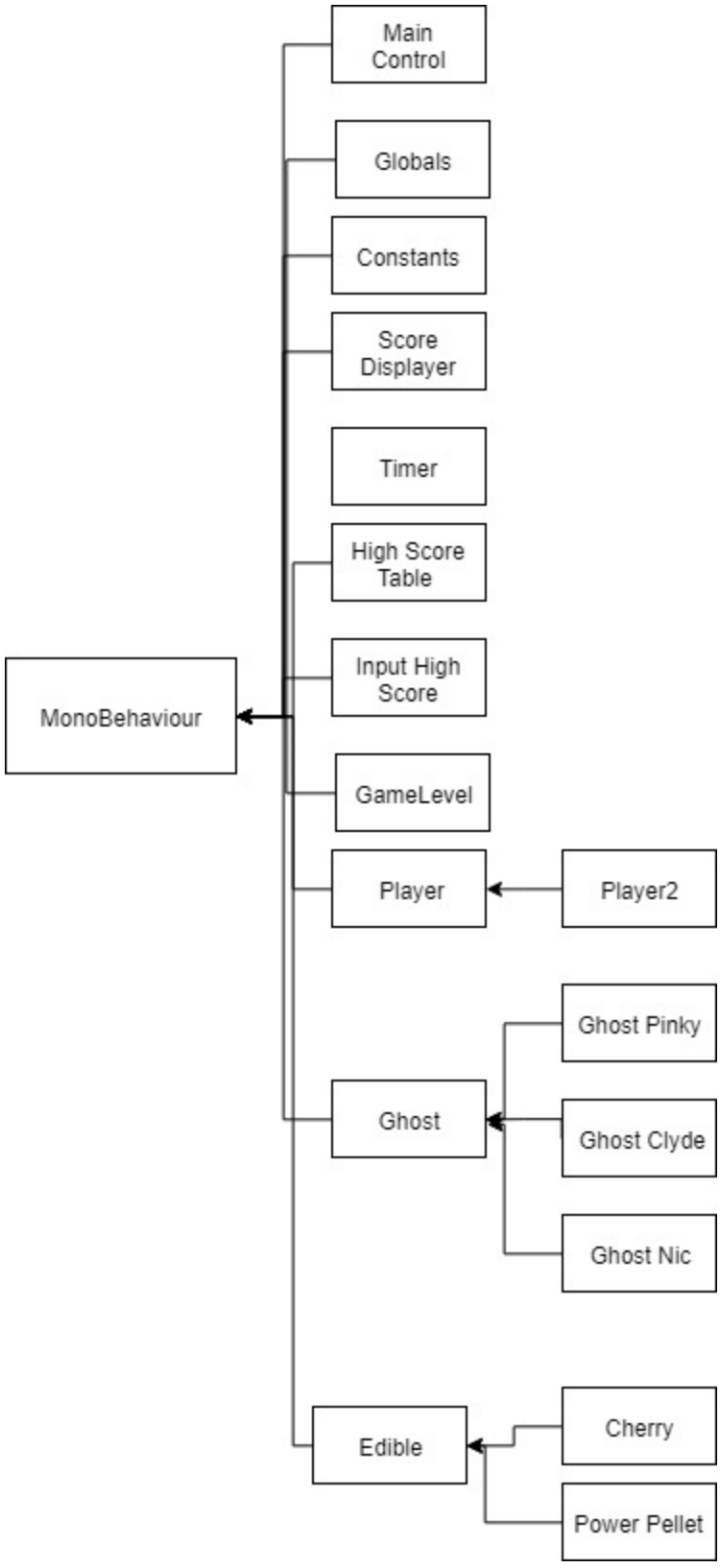
```
public enum Modes {normalPlay, powerPellet, endLevel, gameOver };
```

A finite state machine is used to manage the different states of the game and coordinate the behaviors of the ghosts and players. The states are based on the different modes of during the Game.



2.5 OOP class designs

The diagram shows the high level relationships between the classes of the game. Below I have listed the methods and attributes for each of the classes. The class titles explicitly state when one class inherits from another class. Any polymorphism is shown through the use of 'virtual' and 'override' key words for the relevant methods that it applies to. Note that: this requires the method of the base class to be 'protected' (similar to public however can only be changed and used by inherited classes) instead of private. Where appropriate attributes and methods are private to encapsulate them within the class. i.e. they are not to be used by any external objects. These are indicated by the key word 'private'


Main Control
Globals
Constants
Score Displayer
Timer
High Score Table
Input High Score
MonoBehaviour
GameLevel
Player
Player2
Ghost Pinky
Ghost
Ghost Clyde
Ghost Nic
Edible
Cherry
Power Pellet

2.5.1 Class Player

Controls the movement of the player based on the user's input. Checks for collisions with edibles and walls and applies the appropriate physics. Also keeps track of its lives and whether the player is dead or not.

Attributes:

Protected	playerID : intger
public	playerSpeed : float
private	originalPosition : vector2
private	bc : BoxCollider2D
private	rb : Rigidbody2D

Methods:

Private	Awake
Protected	Virtual Start
Protected	Virtual Fixedupdate
public	GoHome
private	DeadCheck
private	OnCollisionEnter2D

2.5.2 Class player2 Inherits Player

Methods:

Override	Start ()
Override	FixedUpdate()

2.5.3 Ghost

Uses the ghost chase algorithm to attack and run away from players. Effects the players score or lives if a player is hit by a ghost - depending on which mode it's in. Also flashes the ghost sprite's color when in power pellet mode. Finally, it calculates where to respawn the ghost to when the ghost is eaten.

Attributes:

Private	eatenValue : intger
protected	originalPosition : Vector2
public	ghostSpeed : float
private	step : float
private	toggle : bool
private	directions {up, down, left, right} : enum
public	moveProbability : int

public	ghostingProbability : int
private	bc : BoxCollider2D
private	rb : Rigidbody2D
protected	spriteRendered : SpriteRenderer
protected	originalColour : Color

Methods:

Private	Awake
protected	virtual Start
Private	FixedUpdate
protected	GoHome
public	RandomRespawn
private	OnCollisionEnter2D
Private	GetTargetPosition
private	GetDeltaPosition
private	DistanceBetween
private	GetBestDirection
private	MoveInDirection
private	MoveGhost

2.5.4 Ghost Pinky Inherits Ghost

Methods:

Override	Start

2.5.5 Ghost Clyde Inherits Ghost

The speed of Clyde randomly varies between a slow and fast speed.

Methods:

private	Override Start
private	Update

2.5.6 Ghost Nic Inherits Ghost

Methods:

Override	Start

2.5.7 Edible

Detects collisions with players and increases the score of the player when they collide.

Attributes:

Protected	scoreValue: int
public	eaten: bool
private	bc: BoxCollider
private	rb: Rigidbody

Methods:

Protected	virtual Start
protected	virtual SetScoreValue
protecetd	virtual OnTriggerEnter2D

2.5.8 Cherry Inherits Edible

Inherits most of the same things from edible expect: cherry randomly respawns every 10 seconds all over the maze. For 10 seconds it will appear (until it's eaten) then for 10 seconds it will disappear. It reappears into a new random positon each time.

Methods:

Public	Override SetScoreValue
protected	Override Start
public	Override OnTriggerEnter2D
public	CherryPosition

2.5.9 PowerPellet Inherits Edible

Methods:

Public	Override SetScoreValue
public	Override OnTriggerEnter2D

2.5.10 HighScoreTable

Reads high score values from the file and stores it into a list. Then inserts the new high score into the correct position into the list. Finally writes the new sorted list back into the file. This class is also responsible for displaying the high score table when the users wants to view it.

Attributes:

Private	numberOfHighScores : int
Private	numberOfEntries : int
Private	lowestHighScore : int
public static	linesOfTable : string[]

public static	highScoreTableArry : string[,]
private	guiStyleHeadings : GUIStyle
private	guiStyleScores : GUIStyle

Methods:

Private	Start
Private	Update
Private	ReadTableFromFile
Private	SortTable
Private	WriteTableToFile
Private	OnGUI

2.5.11 InputHighScore

Allows the players to enter their team name (with a character limit)

Attributes:

Public	TheInputField : InputField
public	userInput : string

Methods:

Public	Start
Private	AsigningUserInput

2.5.12 Game Level

Controls the requirements from a new level and end of game. When all the edibles have been eaten it will progress the game onto a new level. As the levels progress the difficulty will increase (more ghost and harder mazes). When requires a new level will randomly generates a new (de-looped version) of a maze and re-instantiate all the edibles. When reaches the end of a game because all lives of the both players have been lost it will check if eligible for the High score table; will cause the appropriate change of scene.

Attributes:

private	wallsMatrixH :int[,]
private	wallsMatrixV: int[,]
private	nodesMatrix : int[,,]
private	nodeVisitCountMatrix : int[,]
private	pathTraversalCountMatrix: int[,,]

public	edibleGameObjcet: GameObject
public	powerPelletGameObject: GameObject
public	wallElementGameObject: GameObject
private	Players : GameObject[]
private	ghostGameObject : GameObject
private	ghostPinkyGameObject : GameObject
private	ghostClydeGameObject : GameObject
private	ghostNicGameObject : GameObject
private	scoreDisplayerGameoject : GameObject
private	allEdibleGameObjcets : GameObject[]
private	allWallsGameObjects : GameObject[]
private	whereX : Vector2
private	wallPositionH : Vector2
private	wallPositionV : Vector2
private	randomNumber : System.Random
private	wallProbability : int
private	iStartNode: int
private	jStartNode : int

Methods:

private	Start
private	CreateEdibles
private	ReActivateAllEdibles
private	FreshLevelWalls
private	CleanWalls
private	NodeMatrixCreator
private	DEBUGWriteToFileNodesMatrix
private	DEBUGWriteToFileWallsMatrix
private	DeLoopInator
private	IsNodeAlive
private	StartNodeFinder
private	ChooseTraversal
private	NextClockwise
private	UsePathToFindNode
private	CheckifVistedNode
private	DeleteWallElement
private	Update
private	NewLevel

2.5.13 Timer

Keeps track of the timer count down for both the cherry respawn and power pellet modes

Attributes:

Public	timerLength : float
Public	timerLengthCherry : float

public	coutnDown : bool
public	CherryGameobject : GameObject

Methods:

Private	Start
Private	Update
Public	SetTimer
Private	OnGUI
Private	CherryTimer

2.5.14 Globals

Does not inherit form MonoBehaviour. This class contains a number of variables that are shared across multiple objects. These are very unambiguous static items.

I am aware that Globals are normally considered bad programming practice, but I ensured to the number of global were kept to a minimum and I would only make them Global if I knew they were going to be referenced over multiple different scripts. Making them global made them much easier to access when needed. Also these variables were necessarily only associated with one class e.g. score required to accessed by Player, Ghost, Edibles, Highscore, gameLevel.

Attributes:

Private static	lowestHighScore: int
Private static	numberOfHighScores: int
private static	teamName: string
public static	TeamName: string
private static	score : int
private static	levelNumber: int
private static	numberOfPlayerAlive: int
public static	NumberOfLivesLeft : int[3]
public	Modes {normalPlay, powerPellet, endLevel, gameOver } : enum
private static	mode : Modes
private static	numberOfEdiblesLeft :int

Methods:

Public static	dEBUG
Public static	TeamName
Public static	LowestHighScore
Public static	NumberOfHighScores
Public static	Score
Public static	LevelNumber
Public static	NumberOfPlayerAlive

Public static	Mode
Public static	NumberOfEdiblesLeft

2.5.15 Constants

Stores all the immutable values that are found in multiple places. The benefit of having them in this class is that they're all in one place making it easier to then resize anything all in one go. Did consider encapsulation for certain local variable would be better but just left those to be declared locally or left as literal values. E.g. decided the keep up, right, left, down to 0,1,2,3 because I knew that I wasn't going to change their assigned values and thought it would look too clunky to put names in – no benefit.

Attributes:

public const	eatenValueMedium: int
public const	eatenValueHigh: int
public const	eatenValueMax: int
public const	speedMin : float
public const	speedSlow: float
ublic const	speedFast : float
public const	speedMax : float
public const	initialWallProbability : int
public const	ghostMoveProbability : int
public const	xLeft : int
public const	xRight: int
public const	yTop : int
public const	yBottom : int
public const	centre : int
public const	xShift : int
public const	yShift : int
public const	mazeWidth : int
public const	mazeHeight : int
public const	howManyEdbliesLeftBeforeMovingToNextLevel : int
public const	highScoreFilePath : string

2.5.16 Main control

Attached to the main control scene where the user decides whether they want to play a game or view the high score table. The class is required to read in the HighScore table file for the first time when the game is first played. So that when the game is played for the first time in that session there are high scores that can be compared to the score achieved by the players in order to determine whether they are eligible to enter the high score table.

Attributes:

Private	numberOfHighScores : int
Private	numberOfEntries : int
Private	lowestHighScore : int
public static	linesOfTable : string[]
public static	highScoreTableArry : string[,]

Methods:

Private	Start
Private	ReadTableFromFile

2.5.17 Score displayer

Displays the: score, lives, number of edibles left, level at the bottom of the screen when the users are playing the game. Also displays a transition message between the scene transitions for a new level and game over.

Attributes:

private	guiStyle : GUIStyle

Methods:

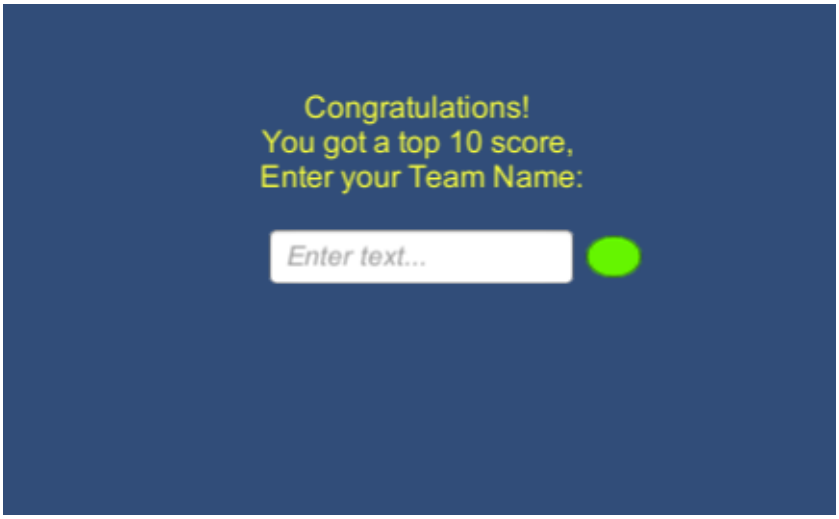
private	start
private	OnGUI

2.6 Interface design

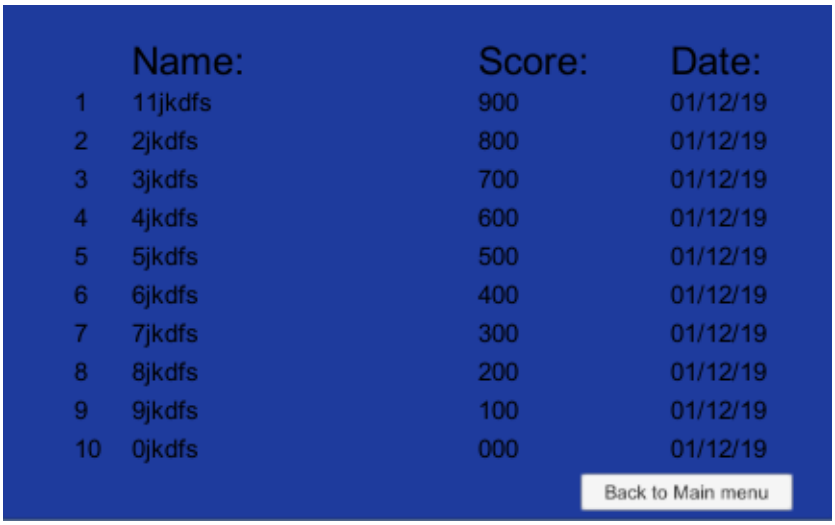
Main control scene: The player can click on the either of the x2 buttons to take them to the next scene they wish to see.

Game frame with levels scene: this is the scene where the users actually play the game and progress though the levels until the game is over. The user has interaction/control over the movement of the player using the arrow keys and awsd. The rest of the functionality is automated by the program I've written.

High score input scene: this scene allows the user to type in a team name. It is presented to the user when the game is over – if they have managed to get a top ten score. They enter the name they want to be remember as and to submit it (and move to the high score view scene) all they are required to do it click the green button.



High score display scene: this scene display the contest of the high score table file, all the user can do is press the back to main menu button.



2.7 Validation & Exception handling

There are not many instances in my code where the users is required to input values in to the program. The only input that the users have control over are: which option they choose in at the main menu, the arrow keys for moving their player; entering in their team name and choosing the option to leave the high scores table. The only one of these that requires any validation is the team name – which is a string.. The validation of this is to limit the number of characters that can be entered. This is done in the definition of

the text entry GUI element. The scene changes are handled by unity, and there isn't really validation required for mouse clicks.

I didn't think it necessary to validate every input parameter of every function. And didn't think it was practical to put an exception handling around every single independent object running in the unity engine. However I did include some exception handling around the file handing and the maze generator algorithm.

The strategy for the handling exceptions raised from file handling was to treat its whole contents as invalid and create a new empty file. The reason behind this was that the format was very strict. If a single part didn't quite work then it implied corruption or tampering and so couldn't be trusted. I would have liked to try and fix/restore the file instead of deleting its whole contents but you could never really be sure how or to what extent the file was damaged. So the best solution was to just to delete it all. Examples of the types of cases that I handle are: the score is not of integer data type, or the file has more than 10 lines.

The second bit of status validation that I included was for the inside GameLevel when it's de-looping the maze. I had used "while" loops to traverse the variable sized trees and wanted to prevent any infinite looping happening if the algorithm went wrong and the exit condition was never met. So I kept a counter of how many trees made up the maze and how many nodes had been visited in each tree. If these counters reached an unrealistically high number then the code throws an exception to indicate that something has gone wrong with the algorithm. This exception is caught in the main menu scene and gives the game an opportunity to try and re- generate the maze again. I calculated the upper bound of the 'unrealistically high numbers' by altering the maze density and testing how many nodes it would have required to traverse if there was full density and then again with the max number of individual trees. I then added an extra precautionary exception handler to catch anything that went wrong whilst it was trying to create a new level, where it would similarly return to the Main control scene.

2.8 Prototyping Feedback

When I had the core of the game working reliably as a prototype, I asked the girls (Emma and Thea) to have a go. They made two suggestion to improve the game that I had not considered:

- Make the ghosts flash when they're in power pellet mode. Would make it much easier to then know when the mode is active; easier to read than the timer in the bottom left corner.
- Don't let the ghost respawn back to the same position after being eaten. Otherwise too easy to take advantage of by sitting in the ghost's home and just keep on eating it.

I incorporated this into the final product.

3 TECHNICAL SOLUTION

3.1 Unity

Unity is a powerful game development application, which is used in the development of many commercial games but is also available free to use by students and non-commercial developers. It provides sophisticated capabilities for full 3D graphics and the simulation of physical interactions between objects (as well as a 2D platform environment, which I what I have used in my project). Unity provides a C# programming interface which can be used to program the properties, behaviors and interaction between the game objects.

For this project only a very limited specific set of the core Unity functionality is used - essentially just enough to provide the basic framework for a game:

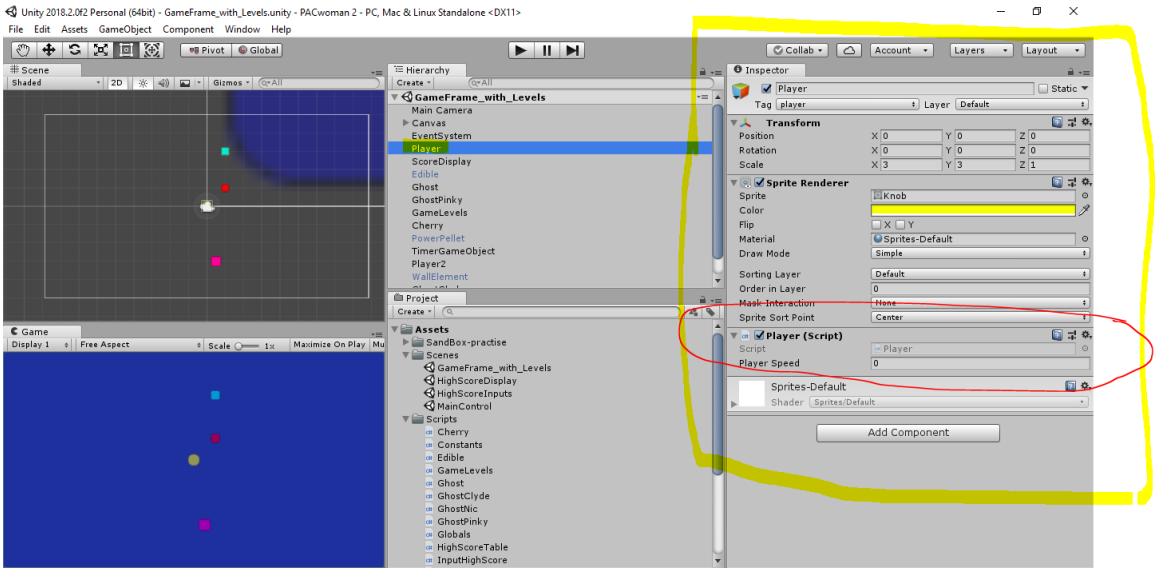
- Game Engine for simple 2D object interactions
- A simple graphical display of the game objects

3.1.1 Overview of Unity features used

Development Environment:

Unity provides a graphical programming interface in which the different game elements can be drawn and given key functional and presentational features. This also allows the C# program scripts to be attached to the graphical elements as one of its 'components'.

e.g. I have opened the Player gameObject and it has the components: Transform, Sprite Renderer and a Script (circled in red) also called player.



It is through these C# scripts that the functionality of the game is programmed.

Overview of the basic Unity elements that are used in my program:

The Unity Game Engine:

The Unity Game Engine provides the scheduler for game actions and events. Through the scheduler the game will be able to interact between the dynamic objects.

The game engine has an Initialisation phase, that is run at the start every time the play button is first pressed, used to set up the game objects. Once the game has stated the game engine then proceeds with a fixed frame rate. Where at each frame it:

- Assesses any resulting events such as collisions and triggers. Performs the corresponding OnTrigger() operations on the relevant objects
- Performs the Update() operation for each game object
- Propagates the dynamic motion of all the objects according to the physics applicable (as the objects move it updates the transform component attributes accordingly)

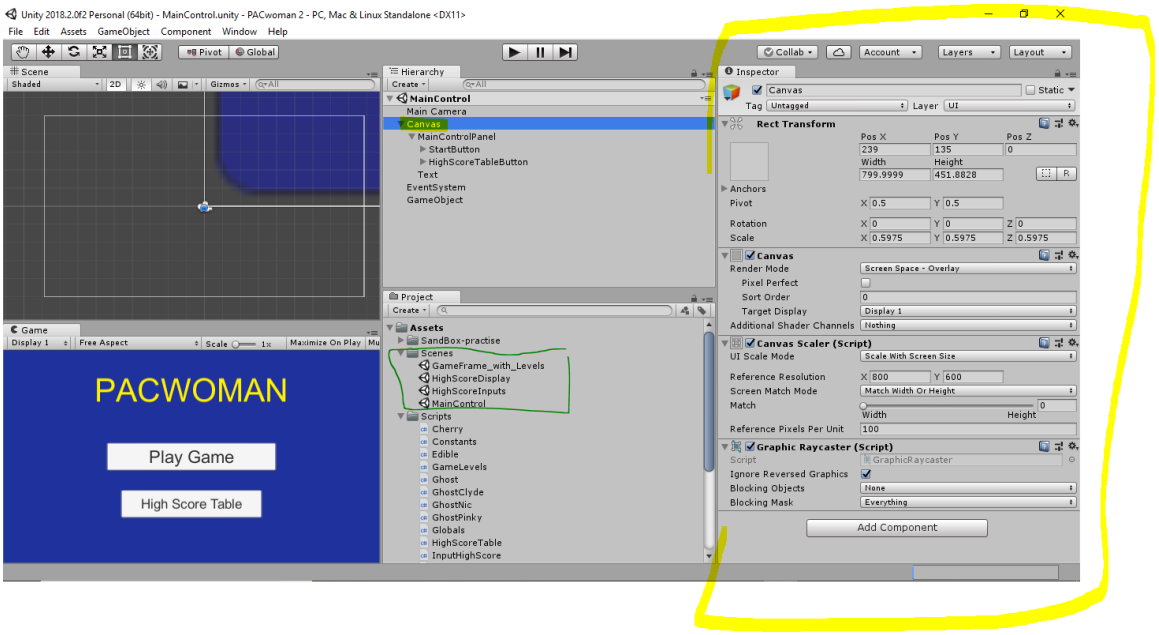
There are a wide range of elements that can be created to populate a game – and which will then interact through the Unity Game Engine.

3.1.2 The Graphical programming environment of unity:

Scene: (almost like the 'background')

Canvas: The space in which the user interface and game elements are presented. The space in which the game plays out. The display/visuals.

An example of a canvas with its componetsis highlighted in yellow.
The list of the different scenes that I use in my game has been circled in green. Currently in the MainControl Scene.

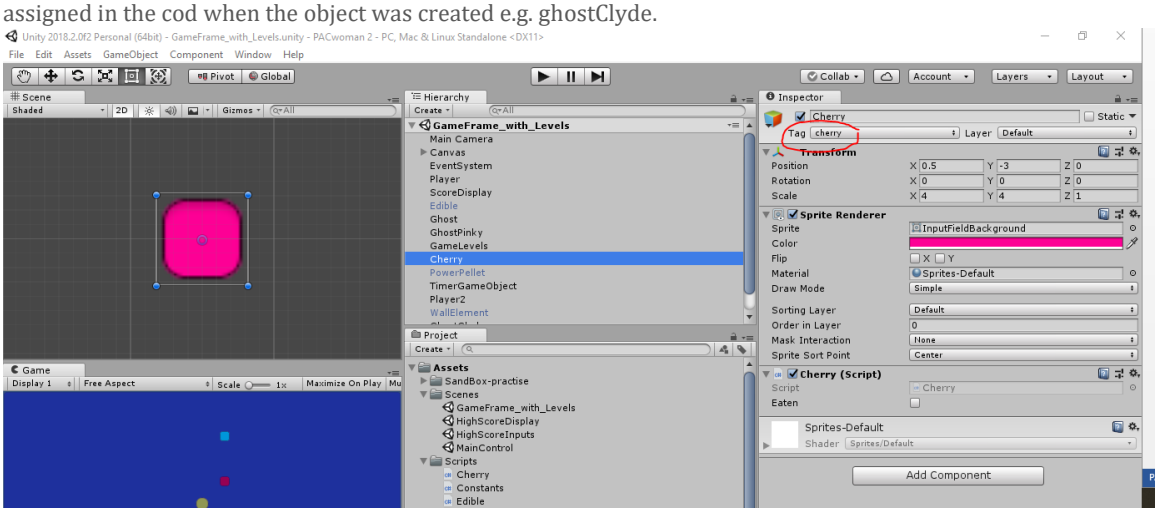


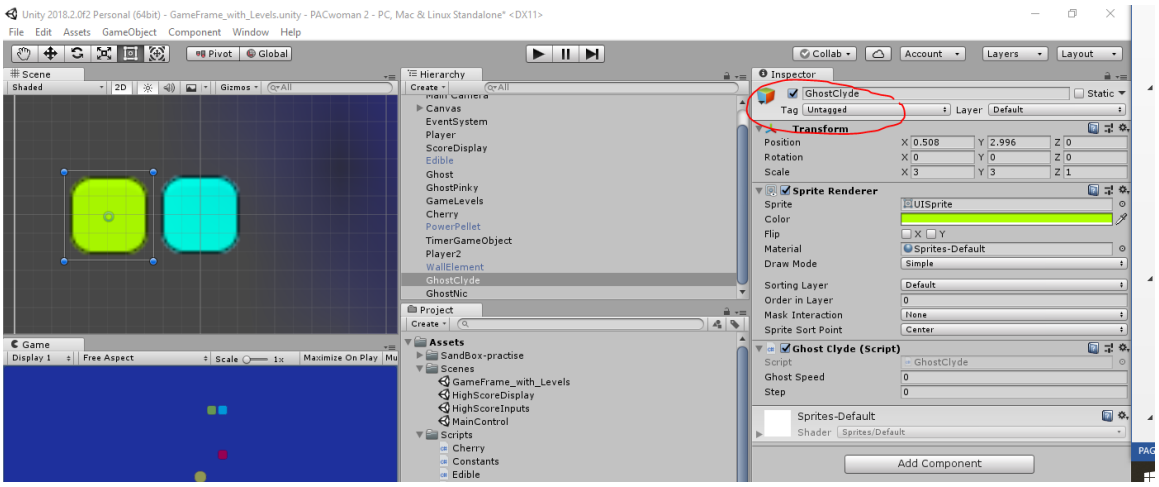
GameObjects: (These are the individual distinct objects that are displayed and interact with each other via the Unity Game Engine. They can have a range of features attached to them)

The following features that are used in this project:

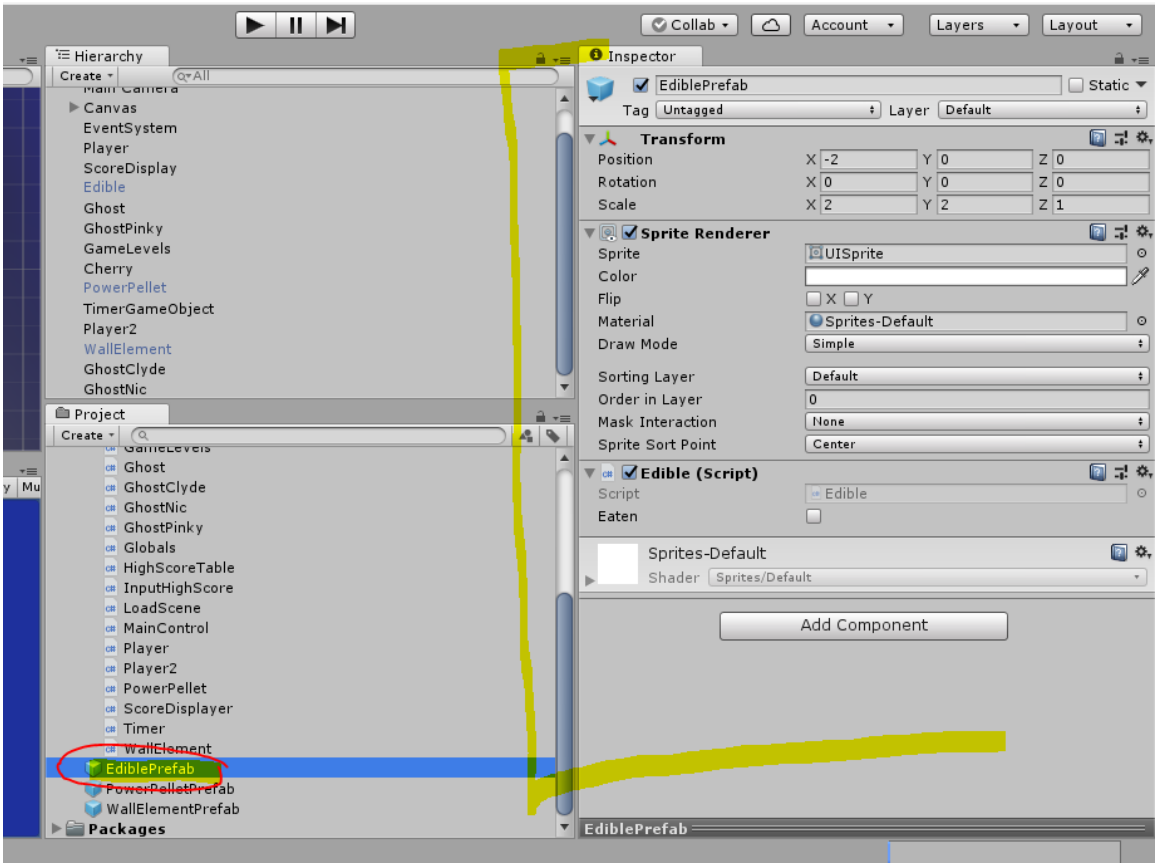
- **Tags:** These allow a string label – a tag – to be associated with an object. The same tag can be associated with many different objects. The Unity Game Engine provides functions which can use a tag to identify a set of game objects (see below).

  Some of the tags I assigned on the untiy graphical side. Whereas some of the tags were assigned in the cod when the object was created e.g. ghostClyde.





- **Components:** (features attached to each GameObject)
  - **Transform** (manages the position, orientation and scale of the game object in the canvas of the scene)
  - **Sprite Rendered** (manages the graphical presentation of the game object on the canvas in the scene – shape, colour – others are left default (material, draw mode, layer, etc)
  - **RigidBody** (Allows the game object to interact via the physics aspects of the game object – essentialy become solid.)
  - **Collider** (allows the object to raise events when colliding with other game objects. These can then be caught and handled in the users program)
  - **C# Script**: attaches the C# program associated with the game object. These provide a means by which you can extend the Unity intrinsic game object class monobehaviour with the specific functionality of your program. The script is used for detecting and handling events, specifying the actions to be taken on start-up of a game object and for every frame.
- **Prefabs:** These provide a means for gathering all of the components of a unity game object into one item which can then be instantiated on/in multiple times to create many identical independent instances in a game.



3.1.3 C# programming scripts in Unity

Each game object that is displayed in a scene and interacts with other game objects via the Unity game engine is based on the Unity class MonoBehaviour.

MonoBehaviour provides a set of handles to which user attaches the particular processing they want in their game. These essentially correspond to messages for the events that the Unity Game Engine will raise. The ones used in this project are:

- Awake()
- Start()
- Update()
- Fixed_Update()
- OnTrigger()

MonoBehaviour also provides operations for manipulating a given game object. The ones used in this project are:

- addComponent
- transform
- others…

Each game object script specifies the class for the object and declares a C# class that inherits from the MonoBehaviour class, to provide it with this integration into the game

engine. Further layers of class hierarchy can be developed through inheritance – e.g. class for a Cherry object can inherited from a more general Edible class which in turn is derived from the MonoBehaviour class.

Details of the full set of Unity functionality used in the project:

Libraries Used (i.e declared in C# "using" statments:

- UnityEngine
- UnityEngine.SceneManagement
- UnityEngine.UI

Components Used

- Transform
- SpriteRenderer
- Script
- Rigidbody2D
- BoxCollider2D
- Camera
- Canvas
- Canvas Scaler
- Graphic Raycaster
- Canvas Renderer
- Image
- EventSystem

3.1.3.1 Features of Unity Classes used

Details of the specific subset of features of each Unity Class that are used in the project:

MonoBehaviour

Is the base class from which every Unity script derives.

Messages

- Awake(): Awake is called when the script instance is being loaded.
- Start(): Start is called on the frame when a script is enabled just before any of the Update methods is called the first time
- Update(): Update is called every frame, if the object is enabled/active.
- FixedUpdate(): This function is called every fixed frame if the object is enabled/active.
- OnCollisionEnter2D(): Sent when an incoming collider makes contact with the Objects collider
- OnTriggerEnter2D(): Sent when another object enters a trigger collider attached to this object
- OnGUI(): OnGUI is called for rendering and handling GUI events.

Inherited Variables:

- enabled
- isActiveAndEnabled:
- gameObject: the gameObject this component is attached to. A component is always attached to a gameObject
- tag: The tag of this game object
- transform: the Transform attached to this game object
- name: the name of this game object

Inherited Public Functions

- GetComponent

Static Functions

- Destroy: Removes a game object, component or asset
- DontDestroyOnLoad: Makes the object target not be destroyed automatically when loading a new scene.
- Instantiate: Returns a copy of the object original

GameObject

Class in UnityEngine / Inherits from: Object / Implemented in UnityEngine.CoreModule
Base class for all entities in Unity Scenes.

Properties

- activeSelf: The local active state of this GameObject (readonly)
- tag: The tag of this game object
- transform: The Transform attached to this GameObject

Public Methods

- AddComponent() Adds a component class named className to the game object
- GetComponent() Returns the component of Type type if the game object has one attached, null if it doesn't.
- SetActive(): Activates/Deactivates the GameObject based on input true/false

Static Methods

- FindGameObjectsWithTag: Returns a list of **active** GameObjetcs tagged with tag. Returns an empty list if no matching GameObject is found.
- FindWithTag: returns one active GameOBject tagged with tag. Returns null if no GameObject is found.

GUI

Class in UnityEngine / Implemented in: UnityEngine.IMGUIModule

The GUI class is the interface for Unity's GUI with manual positioning.

Static Methods

- Label: Make a text or texture label on the screen

GUIStyle

Class in UnityEngine / Implemented in: UnityEngine.IMGUIModule

Styling information for GUI elements.

Most GUI functions accept an optional GUIStyle parameter to override the default style. This allows coloring, fonts and other details to be changed and switched for different states (eg, when the mouse is hovering over the control). Where a consistent look-and-feel is required over a whole GUI design, the GUISkin class is a useful way to collect a set of GUIStyle settings and apply them all at once.

Properties

- fontSize: The font size to use (for dynamic fonts)

UI

Turns a simple label into an interactive input field

Properties

- text: the current value of the input text
- onEndEdit: The Unity Event to call when editing ends. Can add a Listener which is called when the event occurs

SceneManager

Class in UnityEngine.SceneManagement / Implemented in UnityEngine.CoreModule
Scene management at runtime.

Static Methods

- LoadScene: Loads the Scene by its name or index in BuildSettings

C# libraries used:

System.Random

Represents a pseudo-random number generator, which is a device that produces a sequence of numbers that meet certain statistical requirements for randomness.

Methods

- Next: returns a random integer

System.Math

Provides maths functions

Methods

- Sqrt : calculates the square root
- Abs: absolute value, modulus
- Pow: raising to a power

System.IO.File.

Class: Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of FileStream objects.

Methods

- ReadAllLines: Opens a text file, reads all lines of the file into a string array, and then closes the file.

System.IO.StreamWriter

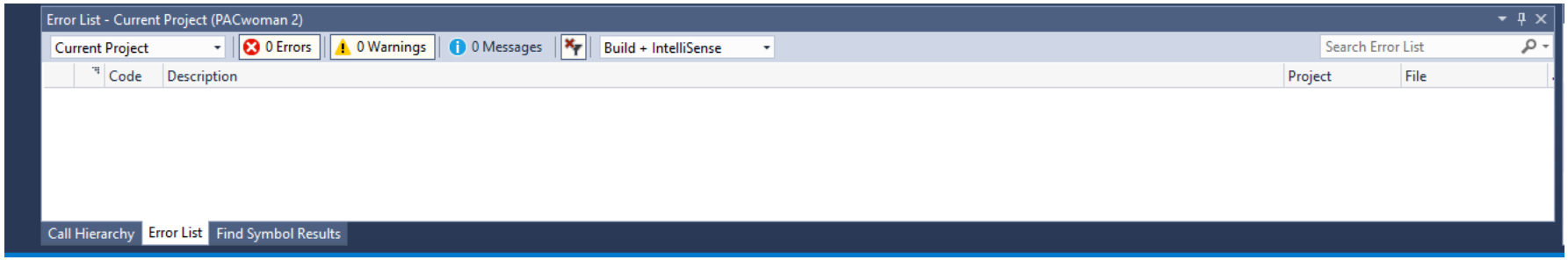
Class: Implements a TextWriter for writing characters to a stream in a particular encoding

Methods

- WriteLine: Writes the provided string as a line to the file

3.2 Complete code listing:

When coding in C# using Visual Studio I ensured that I adhered to the C# coding standard: IDE1006. By address all warnings and messages. Here's evidence: theres no errors, warnings nor messages.



3.2.1 Cherry

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

/*Cherry:  This is a variant of Edible
  its activation is triggered by a Timer so that it appears at random positions for a while then disappears
  It also differs from a standard edible because it doesnt need to be eaten to end a level and it is worth more */
public class Cherry : Edible { //inherits edible class (instead of monobehaviour)

    //declaring a random number
    System.Random randomNumber = new System.Random();

    //the cherry is worth more points than a standard edible
    public override void SetScoreValue()
    {
        scoreValue = Constants.eatenValueHigh;
    }

    //Start: dont use the base function as it has multiple subtle differences from standard Edible:
    /* dont want Cherry to contribute to the number of edibles
```

```
   becasuse game uses the number of edibles left to determine the end of a level
   (i.e. dont want the change in level to be dependent on how many cherries have been eaten) */
    protected override void Start()
    {

        gameObject.tag = "cherry";
        SetScoreValue();
        //not instatiated like the other edibles, so need to activate the gameObject here
        gameObject.SetActive(true);

    }

    //CherryPosition: the cherry's position is randomly generated
    //
    public void CherryPosition()
    {

        transform.position = new Vector2((float)(int)randomNumber.Next(-7, 7), (float)(int)randomNumber.Next(-3, 3)+0.5f);

    }

    //when hit by a player the object is deactivated and the score is increased
    //but unlike base class Edible the NumberOfEdiblesLeft count is not decreased (hence need to override)
    public override void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "player")
        {

            gameObject.SetActive(false);
            Globals.Score += scoreValue;
        }
    }
}
```

3.2.2 Constants

```
public static class Constants
{
    //Eaten Vlaues for edibles:
    public const int eatenValueLow= 1;
    public const int eatenValueMedium = 10;
    public const int eatenValueHigh = 20;
    public const int eatenValueMax = 50;

    //speed for player and ghosts
    public const float speedMin = 1.6f;
    public const float speedSlow = 2.0f;
    public const float speedFast = 4.0f;
    public const float speedMax = 6.0f;

    //lower bound of the maze complexity that still creats a maze
    public const int initialWallProbability = 40; //40
    //the likly hood that the ghost moves in the 'optimal' direction
    public const int ghostMoveProbability = 65;

    //postions on the screen
    public const int xLeft = -7;
    public const int xRight = 7;
    public const int yTop = 4;
    public const int yBottom = -3; //to accommodate the info bar at bottom
    public const int centre = 0;

    //shifting postion to become an index
    public const int xShift = 7;
    public const int yShift = 3;

    //indexing inside of the maze matricies
    public const int mazeWidth = 15;
    public const int mazeHeight = 8;

    //how Many Edblies Left Before can Move To the Next Level
    public const int howManyEdbliesLeftBeforeMovingToNextLevel = 0; //set back to 0 when finish testing (i.e. shortcut)
```

```
    //the path of where the high score table file is stored
    public const string highScoreFilePath = ".\\HighScoreTableFile2.txt"; // at the moment it's;
C:\Users\Lauren\Documents\Unity\PACwoman 2
    //public const string highScoreFilePath = "C:\\Users\\Lauren\\Documents\\NEA2019\\HighScoreTableFile1.txt";
}
```

3.2.3 Edibles

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Edible : MonoBehaviour {

    //the value that the score increases by when eaten
    protected int scoreValue;
    public bool eaten;

    //variables to hold unity components
    private BoxCollider2D bc;
    private Rigidbody2D rb;

    private void Awake()
    {
        //attaching the componenets to the gameobjects
        bc = gameObject.AddComponent<BoxCollider2D>() as BoxCollider2D;
        bc.isTrigger = true; //when true means the objects collide
        rb = gameObject.AddComponent<Rigidbody2D>() as Rigidbody2D;
        //kinematic means that the rigid body is nolonger affected by forces  and effects others when they're collided with
        rb.bodyType = RigidbodyType2D.Kinematic;
    }

    protected virtual void Start ()
    {

        gameObject.tag = "edible"; //attaching a tag to this gameObject
        ++Globals.NumberOfEdiblesLeft;//every time an edible is instatiated the NumberOfEdiblesLeft is increemenetd by one
        SetScoreValue();
    }

    public virtual void SetScoreValue()
    {
        scoreValue = Constants.eatenValueLow;
    }
```

```
    public virtual void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "player")
        {
            // if the object that collides with the edbile has a tag of player then
            //deactivates it, deacrese the  NumberOfEdiblesLeft and increase the players score based on the value of the edible
that it colldied with
            gameObject.SetActive(false);
            --Globals.NumberOfEdiblesLeft;
            Globals.Score += scoreValue;
        }
    }
 }
```

3.2.4 GameLevels

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using System;
using UnityEngine.SceneManagement;

using System.IO;


public class GameLevels : MonoBehaviour {

    private int[,] wallsMatrixH = new int[Constants.mazeWidth, Constants.mazeHeight]; // 15 x 8 (x by y) matrix for the horizintal wall elemnts 0 no
wall & 1 is there is a wall
    private int[,] wallsMatrixV = new int[Constants.mazeWidth, Constants.mazeHeight]; // ditto for vertical walls

    /*nodesMatrix:
     * 16x9 because need an n+1 walls to bound n nodes (need an extra column on the right and an extra row at the bottom)
     * n-n-n requires 3 nodes but 2 walls
     * the z dimension is for the Up = 0 Right = 1 Down = 2 left = 3 walls at each node  (i.e. increasing clocksiwse will work with modulo 3 to loop
round Directions)
    */
    private int[,,] nodesMatrix = new int[Constants.mazeWidth+1, Constants.mazeHeight+1, 4];

    // variables to manage the corrections of closed loops in the maze
    private int[,] nodeVisitCountMatrix = new int[Constants.mazeWidth + 1, Constants.mazeHeight + 1]; // like node mtarix but holding the number of
times each node has been visited
    private int[,,] pathTraversalCountMatrix = new int[Constants.mazeWidth + 1, Constants.mazeHeight + 1, 4]; // like the node matrix but holding the
count of the number of times that each path for a node has been traversed

    //need to be public so that the prefab can be attached to them in the unity window
    public GameObject edibleGameObjcet; //the object for the prefab instantiations
    public GameObject powerPelletGameObject; //public to allow connection to the prefab
    public GameObject wallElementGameObject;

    private GameObject[] Players, ActiveEdibles;
    private GameObject ghostGameObject, ghostPinkyGameObject, ghostClydeGameObject, ghostNicGameObject;
    private GameObject scoreDisplayerGameoject;


    private GameObject[] allEdibleGameObjcets, allWallsGameObjects; //lists of all the edible and wall gameobjects
```

```
//used to positon the edibles and wall elements on the grid when instantiated
private Vector2 whereX;
private Vector2 wallPositionH, wallPositionV;

//declaring a random number
System.Random randomNumber = new System.Random();
private int wallProbability = 0;

void Start() {

    Globals.Mode = Globals.Modes.normalPlay; //initializing the mode

    //just a 90 degree rotation, so can easily turn wall 90 degrees to go from vertical to horizontal walls
    Quaternion rotation90 = Quaternion.Euler(0, 0, 90);

    //creating the outside walls
    wallElementGameObject.tag = "wallOuter";
    wallElementGameObject.transform.localScale = new Vector2(105, 1.5f);
    Instantiate(wallElementGameObject, new Vector2(0, Constants.yTop+1), transform.rotation);
    Instantiate(wallElementGameObject, new Vector2(0, Constants.yBottom - 1), transform.rotation);
    wallElementGameObject.transform.localScale = new Vector2(60, 1.5f);
    Instantiate(wallElementGameObject, new Vector2(Constants.xRight + 1.5f, 0), rotation90);
    Instantiate(wallElementGameObject, new Vector2(Constants.xLeft - 1.5f, 0), rotation90);

    wallElementGameObject.tag = "wall";

    //now instatiates all the edibles
    CreateEdibles();

    //instatiating the walls of the maze for the first time with an initial wall probability
    try { FreshLevelWalls(Constants.initialWallProbability); }
    catch { print("EXCEPTION: failed to generate the maze, returned to MainControl Scene"); SceneManager.LoadScene(0); }

}

void CreateEdibles()
{
    // intatiates all the edibles and powerpellets

    for (int j = Constants.yBottom-1; j <= Constants.yTop; j++)
    {
        for (int i = Constants.xLeft; i <= Constants.xRight; i++)
        {
            whereX = new Vector2(i, j + 0.5f);
            if (Math.Abs(j) == 3 & Math.Abs(i) == 6)
```

```
            {
                //only instantiates the power pellet in the corners of the maze
                Instantiate(powerPelletGameObject, whereX, transform.rotation);
            }
            else
            {
                Instantiate(edibleGameObjcet, whereX, transform.rotation);
            }
        }
    }


}

void ReActivateAllEdibles()
{

    for (int i=0; i < allEdibleGameObjcets.Length; i++) { allEdibleGameObjcets[i].SetActive(true); }

}

/* first populates the Wall matrices randomly
   calculates the node matrix from the wall matrices
   deloopinator: using the node matrix to identify ad remove any closed loops
   instatiates the wall elemenst based on the resulting loop free wall matrices
*/
void FreshLevelWalls(int wallProbability)
{

    // creating the randomly generated maze and storing it in the coresponding wall matrices H and V
    for (int j = Constants.yBottom; j <= Constants.yTop; j++) //8
    {
        for (int i = Constants.xLeft; i <= Constants.xRight; i++) //15
        {
            wallsMatrixH[i + Constants.xShift, j + Constants.yShift] = 0; //shifting postioins into matrix index to name the nodes
            wallsMatrixV[i + Constants.xShift, j + Constants.yShift] = 0;

            //populates the Wall matrices randomly
            if (randomNumber.Next(1, 100) < wallProbability)
            {
                wallsMatrixH[i + Constants.xShift, j + Constants.yShift] = 1;
            }

            if (randomNumber.Next(1, 100) < wallProbability && j < Constants.yTop) //&& so wont do vertivcal bars for the top row
            {
                wallsMatrixV[i + Constants.xShift, j + Constants.yShift] = 1;
```

```
                }

            }
        }
        //calculates the node matrix from the wall matrices
        NodeMatrixCreator();

        //deloopinator: using the node matrix to identify ad remove any closed loops
        DeLoopInator();

        //determines the scale of the maze wall elements that are going to be instantiated
        wallElementGameObject.transform.localScale = new Vector2(7, 1.5f);
        //just a 90 degree rotation, so can easily turn wall 90 degrees to go from vertical to horizontal walls
        Quaternion rotation90 = Quaternion.Euler(0, 0, 90);

        for (int j = Constants.yBottom; j <= Constants.yTop; j++) //8
        {
            for (int i = Constants.xLeft; i <= Constants.xRight; i++) //15
            {
                wallPositionH = new Vector2(i, j);
                wallPositionV = new Vector2(i - 0.5f, j + 0.5f);

                //instatiates the wall elemenst based on the resulting loop free wall matrices
                if (wallsMatrixH[i + Constants.xShift, j + Constants.yShift] == 1)
                {
                    Instantiate(wallElementGameObject, wallPositionH, transform.rotation);
                }

                if (wallsMatrixV[i + Constants.xShift, j + Constants.yShift] == 1) //wont do vertivcal bars for the -3 coordinates i.e bottom row
                {
                    Instantiate(wallElementGameObject, wallPositionV, rotation90);
                }

            }
        }

        if (Globals.DEBUG)
        {
            DEBUGWriteToFileNodesMatrix("NodeMatrixCreator     Nodes   AFTER DELOOPINATOR"); //DEBUG CALLS
            DEBUGWriteToFileWallsMatrix("NodeMatrixCreator     Walls   AFTER DELOOPINATOR");
        }
        }

//destroys all the wall elemenst making up the maze (but not the outter walls)
    void CleanWalls()
    {
```

```
    allWallsGameObjects = GameObject.FindGameObjectsWithTag("wall");
    for (int i = 0; i < allWallsGameObjects.Length; i++) { Destroy(allWallsGameObjects[i]); }

}

/* NodeMatrixCreator:
    uses the H and V wall Matrices to create a new Node matrix which stores the presnt wall elements at each node in the maze
    for each node there are four elements one for each of thw alls that meet at the node
    USING : Up=0 Right=1 Down=2 left=3 i.e. increasing clocksiwse (will work with modulo 3 to loop round the different Directionss)
*/
void NodeMatrixCreator()
{
    for (int j = 0; j < Constants.mazeHeight + 1; j++) //9
    {

        for (int i = 0; i < Constants.mazeWidth + 1; i++) //16
        {
            // inititializing the Count matrcies to 0
            nodeVisitCountMatrix[i, j] = 0;
            for (int k = 0; k <= 3; k++)
            {
                pathTraversalCountMatrix[i, j, k] = 0;
            }

            // populating the nodesMtarix using the wallsMatrix H and V (corners and edges are special and are dealt with seperately)
            if (i != Constants.mazeWidth && j != Constants.mazeHeight && i != 0 && j != 0)
            {
                //BTW: e.g. the Up wall element for one node is the same as the Down wall elemnt for the node above
                nodesMatrix[i, j, 0] = wallsMatrixV[i, j]; //assigning the up to the node
                nodesMatrix[i, j, 1] = wallsMatrixH[i, j]; //assigning the right to the node
                nodesMatrix[i, j, 2] = wallsMatrixV[i, j - 1]; ////assigning the down node
                nodesMatrix[i, j, 3] = wallsMatrixH[i - 1, j]; //assigning the left to the node
            }
            //--------------------EDGE CASES (where meet the outter walls)---------
            else if (i == 0 && (j != 0 && j != Constants.mazeHeight)) //left edge excluding corners
            {
                nodesMatrix[i, j, 0] = wallsMatrixV[i, j]; //assigning the up to the node
                nodesMatrix[i, j, 1] = wallsMatrixH[i, j]; //assigning the right to the node
                nodesMatrix[i, j, 2] = wallsMatrixV[i, j - 1]; ////assigning the down node
            }
            else if (i == Constants.mazeWidth && (j != 0 && j != Constants.mazeHeight)) //right edge excluding corners
            {
                nodesMatrix[i, j, 3] = wallsMatrixH[i - 1, j]; //assigning the left to the node
            }
            else if (j==0 && i!=0 && i!= Constants.mazeWidth) //bottom edge excluing corners
            {
```

```
                nodesMatrix[i, j, 0] = wallsMatrixV[i, j]; //assigning the up to the node
                nodesMatrix[i, j, 1] = wallsMatrixH[i, j]; //assigning the right to the node
                nodesMatrix[i, j, 3] = wallsMatrixH[i - 1, j]; //assigning the left to the node
            }
        }

    }
    //----------------------------CORNER CASES-------------------------------------
    //bottom left corner case
    nodesMatrix[0, 0, 0] = wallsMatrixV[0, 0]; //assigning the up to the node
    nodesMatrix[0, 0, 1] = wallsMatrixH[0, 0]; //assigning the right to the node

    //top left corner case
    nodesMatrix[0, Constants.mazeHeight-1, 1] = wallsMatrixH[0, Constants.mazeHeight - 1]; //assigning the right to the node
    nodesMatrix[0, Constants.mazeHeight - 1, 2] = wallsMatrixV[0, Constants.mazeHeight - 2]; ////assigning the down node

    //bottom right corner case
    nodesMatrix[Constants.mazeWidth, 0, 3] = wallsMatrixH[Constants.mazeWidth-1, 0]; //assigning the left to the node
    //there are no Vertical wall elements at j=15

    //top right corner case
    nodesMatrix[Constants.mazeWidth, Constants.mazeHeight - 1, 3] = wallsMatrixH[Constants.mazeWidth - 1, Constants.mazeHeight - 1]; //assigning
the left to the node
    //there are no Vertical wall elements at j=15

    DEBUGWriteToFileNodesMatrix("NodeMatrixCreator     Nodes  BEFORE DELOOPINATOR"); //DEBUG CALLS
    DEBUGWriteToFileWallsMatrix("NodeMatrixCreator     Walls  BEFORE DELOOPINATOR");
}

// --------------------------DEBUG START-----------------------------------------------------
// DEBUGWriteToFileNodesMatrix: this is a useful pretty print function to show the NodeMatrix as a maze into a text file
//like this:
/*
 NodeMatrixCreator     Nodes
                            top: 8 :
                            mid: 8 :
                            bot: 8 :
                            top: 7 :
                            mid: 7 :     -- -- --    -- --               --
                            bot: 7 : i       i          i     i  i        i
                            top: 6 : i       i          i     i  i        i
                            mid: 6 :                                -- --
                            bot: 6 :            i  i  i     i     i  i        i
                            top: 5 :            i  i  i     i     i  i        i
                            mid: 5 :     --    --          --    --       -- -- --
                            bot: 5 :                 i          i  i     i  i
```

```
top: 4 :                   i           i  i     i  i
mid: 4 :           -- --    -- --       -- --
bot: 4 :             i           i     i  i  i
top: 3 :             i           i     i  i  i
mid: 3 :  -- --          --    --          -- -- -- --
bot: 3 :    i                    i           i     i
top: 2 :    i                    i           i     i
mid: 2 :     --       --        --       --
bot: 2 :                i        i  i             i
top: 1 :                i        i  i             i
mid: 1 :         --           --             --       --
bot: 1 :       i     i     i  i        i        i
top: 0 :       i     i     i  i        i        i
mid: 0 :  --    --       --       --    --          --
bot: 0 :
bot:   :000111222333444555666777888999000111222333444555
```

```
        */

void DEBUGWriteToFileNodesMatrix(string where)
    {

        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Lauren\Documents\NEA2019\WriteLines4.txt", true))
        {
            file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(where);
        }

        for (int j = Constants.mazeHeight; j > -1; j--) //9
        {
            string toprow = "                                                                    top: " + j + " :";
            string midrow = "                                                                    mid: " + j + " :";
            string botrow = "                                                                    bot: " + j + " :";

            for (int i = 0; i < Constants.mazeWidth + 1; i++) //16
            {
                if (nodesMatrix[i, j, 0] == 1) { toprow += " "; toprow += "i"; toprow += " "; } else { toprow += "   "; }
                if (nodesMatrix[i, j, 3] == 1) { midrow += "-"; midrow += " "; } else { midrow += "  "; }
                if (nodesMatrix[i, j, 1] == 1) { midrow += "-"; } else { midrow += " "; }
                if (nodesMatrix[i, j, 2] == 1) { botrow += " "; botrow += "i"; botrow += " "; } else { botrow += "   "; }

            }

            using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Lauren\Documents\NEA2019\WriteLines4.txt", true))
            {
                file.WriteLine(toprow); file.WriteLine(midrow); file.WriteLine(botrow);
            }
```

```
    }
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Lauren\Documents\NEA2019\WriteLines4.txt", true))
    {
        file.WriteLine("                                                                    bot:   :000111222333444555666777888999000111222333444555");
    }
}

// DEBUGWriteToFileWallsMatrixthis :is a useful pretty print function to show the Walls Matrices as a maze into a text file
/*
 * NodeMatrixCreator     Walls
                                V: 7 :
                                H: 7 :    -- -- --    -- --              --
                                V: 6 :|       |          |    | |       |
                                H: 6 :                              -- --
                                V: 5 :           |  |  |    |    |  |       |
                                H: 5 :    --    --         --    --       -- -- --
                                V: 4 :                |          |  |    |  |
                                H: 4 :          -- --    -- --       -- --
                                V: 3 :            |          |    |  |  |
                                H: 3 : -- --          --    --          -- -- -- --
                                V: 2 :   |                   |          |    |
                                H: 2 :    --       --       --       --
                                V: 1 :              |       | |             |
                                H: 1 :       --         --             --       --
                                V: 0 :     |    |    |  |       |       |
                                H: 0 : --    --       --       --    --          --

 *
 */
void DEBUGWriteToFileWallsMatrix(string where)
{
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Lauren\Documents\NEA2019\WriteLines4.txt", true))
    {
        file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(" "); file.WriteLine(where);
    }

    for (int j = Constants.mazeHeight - 1; j > -1; j--) //9
    {
        string H = "                                                                    H: " + j + " :";
        string V = "                                                                    V: " + j + " :";

        for (int i = 0; i < Constants.mazeWidth; i++) //16
        {
            if (wallsMatrixV[i, j] == 1) { V += "|  "; } else { V += "   "; }
            if (wallsMatrixH[i, j] == 1) { H += " --"; } else { H += "   "; }
```

```
            }

            using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Lauren\Documents\NEA2019\WriteLines4.txt", true))
            {
                file.WriteLine(V); file.WriteLine(H);
            }

        }
    }

    void DeLoopInator()
    {

        int iOfNode, jOfNode, iOfNewNode, jOfNewNode;
        int testingCounter1, testingCounter2;//used to make the loops finite, primarily for DEBUG
        int directionOfTraversal, directionCameFromNew;

        //will de-loop-ify for each tree in the maze, when cant find any new trees it will be finished

        //loop over the trees in a maze
        testingCounter2 = 0;
        while (StartNodeFinder(out iStartNode, out jStartNode) == true) //when true means it has found a new tree i.e. a node that has not been visited
before
        {
            ++testingCounter2; //keeping track of the number of trees
            //to avoid infinite looping, if error
            if (testingCounter2 > 50) { Exception ex = new Exception("EXCEPTION: testingcounter2 is bigger than 50 i.e. tried to deloop more than 50
trees"); throw ex; } //created my own exception
            if (Globals.DEBUG) { print("                                                                    start node: Counter2 " + testingCounter2 + "    "
+ iStartNode + ", " + jStartNode); }

            ++ nodeVisitCountMatrix[iStartNode, jStartNode]; // increments the visit count by 1 for the start node of the tree

            iOfNode = iStartNode; //initiliaizing the values for the tree traversal loop
            jOfNode = jStartNode;
            directionOfTraversal = 3;//using 3 because want it to start the traversal coming in from the left

            //loops over the nodes in a tree, until completeTraversal of the tree
            testingCounter1 = 0;//how many nodes in a tree
            //ChooseTraversal: picks a new direction to traverse, returns flase when the whole tree has been traversed
            //NOTE: as well as chousing the node to traverse to, ChooseTraversal also breaks any closed maze loops
            while (ChooseTraversal(iOfNode, jOfNode, directionOfTraversal, out iOfNewNode, out jOfNewNode, out directionCameFromNew) == true)
            {
                ++testingCounter1;
                //to avoid infinite looping, if error
```

```
                if (testingCounter1 > 500) {Exception ex = new Exception("EXCEPTION: testingcounter1 is bigger than 500 i.e. tried to deloop a tree
with morethan 500 nodes"); throw ex ; } //created my own exception
                if (Globals.DEBUG) { print("     DeLoopInator: new node :  " + iOfNewNode + ", " + jOfNewNode); }

                //once has chosen the new node to traverse to, it moves to it i.e. sets it as the current node
                iOfNode = iOfNewNode;
                jOfNode = jOfNewNode;
                directionOfTraversal = directionCameFromNew;
            }

        }

    }

    //IsNodeAlive: not all nodes have wall elements attached to them, this determines wether or not it has any wall elements
    bool IsNodeAlive(int i, int j)
    {
        int numberOfWallElements = 0;
        for (int k = 0; k <= 3; k++)
        {
            numberOfWallElements += nodesMatrix[i, j, k];
        }
        if (numberOfWallElements > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    int iStartNode, jStartNode; // used to hold the start position of the tree thats being worked on
    // StartNodeFinder: finding a node to start on i.e. for a tree that hasnt yet been de-loop-ified
    bool StartNodeFinder(out int iStartNode, out int jStartNode)
    {
        // initialized to an invalid reffernce
        iStartNode = -1;
        jStartNode = -1;

        for (int j = 0; j < Constants.mazeHeight + 1; j++) //9  // loops thourgh all the nodes
        {
            for (int i = 0; i < Constants.mazeWidth + 1; i++) //16
            {
                // checks that the node has walls connected to it AND that has never been visited i.e. start of a tree
                if (IsNodeAlive(i, j) == true && nodeVisitCountMatrix[i, j] == 0)
```

```
            {
                iStartNode = i;
                jStartNode = j;
                return true; // has found a start node
            }

        }


    }
    return false;// has not found a start node, the whole maze has been deloopified
    }

/* ChooseTraversal:
     * chooses a candidate direction
     * finds the corresponding node to the direction
     * checks if its has ever been visited before:
     *      never been visted before=okay to traverse to it
     *
     *
     */

    bool ChooseTraversal (int iOfNode, int jOfNode, int directionCameFrom, out int iOfNewNode, out int jOfNewNode, out int directionCameFromNew)
    {
        int tempDirectionNewpath, numberOfWallElementsAtNode=0;

        if (Globals.DEBUG) { print("          NodeMtarix:  " + iOfNode + ", " + jOfNode + ", " + nodesMatrix[iOfNode, jOfNode, 0] + ", " +
nodesMatrix[iOfNode, jOfNode, 1] + ", " + nodesMatrix[iOfNode, jOfNode, 2] + ", " + nodesMatrix[iOfNode, jOfNode, 3]); }

        tempDirectionNewpath = NextClockwise(iOfNode, jOfNode, directionCameFrom);//candidate direction
        directionCameFromNew = (tempDirectionNewpath + 2) % 4;//based on the candidate direction it works out what its direction came from would be if
it it moved to this node

        if (Globals.DEBUG){ print("          tempDirection:  " + tempDirectionNewpath); }

        UsePathToFindNode(iOfNode, jOfNode, tempDirectionNewpath, out iOfNewNode, out jOfNewNode);//returns candidate node

        //the candidate node have never been visited
        if (CheckifVistedNode(iOfNewNode, jOfNewNode) == false)
        {
            if (Globals.DEBUG) { print("          ENTER succesful traversal to new node"); }

            ++nodeVisitCountMatrix[iOfNewNode, jOfNewNode];
            ++pathTraversalCountMatrix[iOfNode, jOfNode, tempDirectionNewpath];

            if (Globals.DEBUG){ print("          LEAVE succesful traversal to new node"); }
```

```
            return true;
        }

        else //candidate node HAS been visited before
        {
            //      check if tree traversal is complete
            bool alldone = true;
            for (int k = 0; k <= 3; k++)
            {
                if (pathTraversalCountMatrix[iOfNewNode, jOfNewNode, k] == 0 && nodesMatrix[iOfNewNode, jOfNewNode, k] > 0) { alldone = false; }
            }
            if (alldone)
            {
                if (Globals.DEBUG){ print("                                    all done"); }
                return false;
            }

            //calculates the number of wall elemenets at the node
            for (int k =0; k <= 3; k++) { numberOfWallElementsAtNode += nodesMatrix[iOfNewNode, jOfNewNode, k]; }
            //if back at strat node and allls its wall elements have been visited then
            if (iOfNewNode == iStartNode && jOfNewNode == jStartNode && nodeVisitCountMatrix[iOfNewNode, jOfNewNode] == numberOfWallElementsAtNode)
            {
                if (Globals.DEBUG){ print("                                                ERROR all done"); }
                // finished the tree
                return false;
            }


            //checks if the path its about to traverse has already been traversed, but in the opposite direction
            //this case is NOT A LOOP
            if (CheckifVistedNode(iOfNewNode, jOfNewNode) == true && pathTraversalCountMatrix[iOfNewNode, jOfNewNode, (tempDirectionNewpath + 2) % 4]
== 1)
            {
                ++nodeVisitCountMatrix[iOfNewNode, jOfNewNode];
                ++pathTraversalCountMatrix[iOfNode, jOfNode, tempDirectionNewpath];
                return true;
            }

            // the node has been visited and the path has never been traversed
            //this case IS A LOOP, need deloopifying
            else if (CheckifVistedNode(iOfNewNode, jOfNewNode) == true && pathTraversalCountMatrix[iOfNewNode, jOfNewNode, (tempDirectionNewpath + 2) %
4] == 0)
            {
                //deletes the connecting wall element that closes the loop
                DeleteWallElement(iOfNode, jOfNode, tempDirectionNewpath, iOfNewNode,jOfNewNode);
```

```
                if (Globals.DEBUG){ print("
Wall has been deleted " + iOfNode + " " + jOfNode + " " + tempDirectionNewpath); }
                iOfNewNode = iOfNode;
                jOfNewNode = jOfNode;
                directionCameFromNew = directionCameFrom;
                return true;
            }

        }

        return false;

    }

    //calculates the next clockwise direction from the one you came in on
    int NextClockwise(int iOfNode, int jOfNode, int directionCameFrom)
    {
        int directionGoingTo;
        for (int a = 1; a <= 4; a++)
        {

            directionGoingTo = (directionCameFrom + a) % 4;
            if (nodesMatrix[iOfNode, jOfNode, directionGoingTo] == 1)
            {
                return directionGoingTo;
            }
        }

        //should never get here because would have checkecked that the node is alive first, will always have at least one path to go down even if its
the one it just came from
        return directionCameFrom;
    }

    void UsePathToFindNode(int iOfNode, int jOfNode, int direction, out int iOfNewNode, out int jOfNewNode)
    {
        int i = -1;
        int j = -1;
        if(direction == 0) //up
        {
            i = iOfNode;
            j = jOfNode + 1;
        }

        else if (direction == 1)//right
        {
```

```
        i = iOfNode + 1;
        j = jOfNode;
    }

    else if (direction == 2)//down
    {
        i = iOfNode;
        j= jOfNode - 1;
    }

    else if (direction == 3)//left
    {
        i = iOfNode - 1;
        j = jOfNode;
    }
    iOfNewNode = i;
    jOfNewNode = j;
}

bool CheckifVistedNode(int iOfNode, int jOfNode) //returns false ONLY if has never been visited
{
   if( nodeVisitCountMatrix[iOfNode, jOfNode] == 0)
    {
        return false;
    }
    return true;
}


// DeleteWallElement; removes the wall element from all the matrices
void DeleteWallElement(int iOfNode, int jOfNode, int DirectionPath, int iOfNewNode, int jOfNewNode)
{
    pathTraversalCountMatrix[iOfNode, jOfNode, DirectionPath] = 0;
    pathTraversalCountMatrix[iOfNewNode, jOfNewNode, (DirectionPath + 2) % 4] = 0;

    nodesMatrix[iOfNode, jOfNode, DirectionPath] = 0;
    nodesMatrix[iOfNewNode, jOfNewNode, (DirectionPath + 2) % 4] = 0;

    if (DirectionPath == 0)
    {
        wallsMatrixV[iOfNode, jOfNode] = 0;//removing the Up wall element from the node
    }
    if (DirectionPath == 1)
    {
        wallsMatrixH[iOfNode, jOfNode] = 0;//removing the Right wall element from the node
    }
```

```
        if (DirectionPath == 2)
        {
            wallsMatrixV[iOfNode, jOfNode-1] = 0;//removing the Down wall element from the node
        }
        if (DirectionPath == 3)
        {
            wallsMatrixH[iOfNode-1, jOfNode] = 0;//removing the Left wall element from the node
        }

        }
bool neverbefore = true;
    int wait = 0;

    void Update () {
        //first time through it attaches all the other gameobjects it needs (from different classes): edibles & ghosts
        ActiveEdibles = GameObject.FindGameObjectsWithTag("edible");
        if (neverbefore)
        {
            neverbefore = false;
            // asigning all the edibles to this list of gameobjects
            allEdibleGameObjcets = GameObject.FindGameObjectsWithTag("edible");

            //asigning all the ghosts to gameobjects
            ghostGameObject = GameObject.FindGameObjectWithTag("ghost");
            ghostPinkyGameObject= GameObject.FindGameObjectWithTag("ghostPinky");
            ghostClydeGameObject = GameObject.FindGameObjectWithTag("ghostClyde");
            ghostNicGameObject = GameObject.FindGameObjectWithTag("ghostNic");

            //at the start of the game, first level, only want one ghost active
            ghostPinkyGameObject.SetActive(false);
            ghostClydeGameObject.SetActive(false);
            ghostNicGameObject.SetActive(false);

        }

        //-----------------MANAGING LEVELS AND END OF GAME------------------
        if (Globals.Mode == Globals.Modes.gameOver)
        {
            ++wait;
            //waits a couple of seconds to display end of game message before returing to the main control
            if (wait > 250)
            {
                //deactivating the GUI text
                scoreDisplayerGameoject = GameObject.FindGameObjectWithTag("scoreDisplayer");
                scoreDisplayerGameoject.SetActive(false);
```

```
            //checking it the Score qaulifies for the High SCore table:
            //if thers isnt 10 values in high score table yet, you qualify anway
            if (Globals.NumberOfHighScores < 10) { SceneManager.LoadScene(3); }
            // (there are 10 values already in the tabel)
            //and if thier score is high enough i.e. eligible for a high score, then takes the user to the Input high score Scene
            else if (Globals.LowestHighScore < Globals.Score) { SceneManager.LoadScene(3); }
            //if not eligible to get onto the High Score Table then returns the user to the Main control Scene
            else { SceneManager.LoadScene(0); }
        }
    }
    //only game over when both the players have lost all thier lives
    else if (Globals.NumberOfLivesLeft[1] == 0 && Globals.NumberOfLivesLeft[2] == 0 )
    {
        Globals.Mode = Globals.Modes.gameOver;

    }

    else if (Globals.Mode == Globals.Modes.endLevel)
    {
        ++wait;
        if (wait > 80)
        {
            wait = 0;
            try { NewLevel(); }
            catch { print("EXCEPTION: failed to generate the a new level, returned to MainControl Scene"); SceneManager.LoadScene(0); }
        }
    }
    //once eaten all the edibles (including powerpellets) can porgress to the next level
    else if (ActiveEdibles.Length <= Constants.howManyEdbliesLeftBeforeMovingToNextLevel)
    {
        Globals.Mode = Globals.Modes.endLevel;

    }
}

/*NewLevel:
 * moves the player back to home position
 * reactivates all the Edibles
 * regenerates the walls of the maze
 * adds new ghosts depending on the level
 */
void NewLevel()
{
    //moves the player back to home position
    Players = GameObject.FindGameObjectsWithTag("player");
```

```
        for (int i = 0; i < Players.Length; i++) { Players[i].GetComponent<Player>().GoHome(); }

        //reactivates all the Edibles

        ReActivateAllEdibles();
        ActiveEdibles = GameObject.FindGameObjectsWithTag("edible");
        Globals.NumberOfEdiblesLeft = ActiveEdibles.Length;

        //regenerates the walls of the maze
        CleanWalls();
        //as the levels gets higher,  increases the density of the wall elements in the maze i.e. making the maze harder
        wallProbability = 40 + (int)(Globals.LevelNumber * 0.8);
        //doesnt let the maze get more dense than 70 though because the maze is no longer fun
        if (wallProbability > 70) { wallProbability = 70; }
        FreshLevelWalls(wallProbability);

        ++Globals.LevelNumber;

        //as level progresses increase he number of ghosts
        ghostGameObject.GetComponent<Ghost>().RandomRespawn(); // at each new level the ghost randomly respawns to a postion
        if (Globals.LevelNumber>=0) { ghostPinkyGameObject.SetActive(true); ghostPinkyGameObject.GetComponent<Ghost>().RandomRespawn(); }
        if (Globals.LevelNumber >=3) { ghostClydeGameObject.SetActive(true); ghostClydeGameObject.GetComponent<Ghost>().RandomRespawn(); }
        if (Globals.LevelNumber >=5) { ghostNicGameObject.SetActive(true); ghostNicGameObject.GetComponent<Ghost>().RandomRespawn(); }

        Globals.Mode = Globals.Modes.normalPlay; //sets it back to normal play once completed all the editing for the new level

    }

}
```

3.2.5 Ghost

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class Ghost : MonoBehaviour {

    private int eatenValue, counter = 0;

    public float ghostSpeed, step;
    private bool toggle = true;

    protected Vector2 originalPosition;

    //creating an enum used to define directiosn of movement
    private enum Directions {up, down, left, right};

    //allows for control over what the sprite looks like
    protected SpriteRenderer spriteRenderer;
    protected Color originalColour;

    //variables to hold unity components
    private BoxCollider2D bc;
    private Rigidbody2D rb;

    //defining a random number
    System.Random randomNumber = new System.Random();

    private void Awake()
    {
        //attaching the components to gameobject
        bc = gameObject.AddComponent<BoxCollider2D>() as BoxCollider2D;
        bc.isTrigger = false;
        rb = gameObject.AddComponent<Rigidbody2D>() as Rigidbody2D;
        rb.bodyType = RigidbodyType2D.Dynamic;
```

```
        step = 1;
        //prevents the gameobject lousing the correct sense of direction, otherwise the gameobject will rotate when caught on
corners
        rb.freezeRotation = true;
        //attaching the renderre of the the sprite  to  'spriteRenderer'
        spriteRenderer = gameObject.GetComponent<SpriteRenderer>();
    }

    // Use this for initialization
    protected virtual void Start () {

        gameObject.tag = "ghost";
        eatenValue = Constants.eatenValueMax;
        ghostSpeed = Constants.speedSlow;
        originalPosition = new Vector2(Constants.xRight, Constants.yTop);
        GoHome(); //at the start of the game the ghosts begin @ thier original position
        originalColour = spriteRenderer.color;//initializing the colour of the sprite

    }

    // Update is called once per frame
    void FixedUpdate ()
    {

        MoveGhost();
        if (Globals.Mode == Globals.Modes.powerPellet)
        {
            //making the ghost 'flash' by toggeling the colour of the sprite
            counter++;
            if (counter == 10) { toggle = !toggle;  counter = 0; }
            if (toggle) { print("                              toogle true"); spriteRenderer.color = Color.white; }
            if (!toggle) { print("                               toogle false"); spriteRenderer.color = originalColour; }

        }
        else { spriteRenderer.color = originalColour; }


        //once at a high enough level the ghosts travel really fast
```

```
        if(Globals.LevelNumber > 6) { ghostSpeed = Constants.speedFast; }

    }

    Directions bestDirection; // declared here so not reset everytime MoveGhost is called
    int count = 0;
    int randomWait = 15;

    void MoveGhost()
    {

        ++count;
        if (count == randomWait)
        {
            //only works out a new best direction every 10frames, gives it a chance get somewhere different before changing
direction

            count = 0;
            randomWait = randomNumber.Next(10, 25);

            GameObject[] players; //list which holds the x2 player gameobjects
            players = GameObject.FindGameObjectsWithTag("player");
            //defining the positions locally becuase want it to be redeclared every frame
            Vector2 targetPosition, trialPosition;

            //stores the x2 diffrent distances of the players from the ghost
            float[,] distances = new float[Enum.GetValues(typeof(Directions)).Length, 2]; // 2D array with size of first index
4 (the size/length is 4: up down left right) second index size 2 for the number of players

            for (int i = 0; i <= Globals.NumberOfPlayerAlive; i++)
            {
                targetPosition = GetTargetPosition(players[i]);//calculates the tagret positon seperatley for each player

                foreach (Directions dir in Enum.GetValues(typeof(Directions)))
                {
                    //for each direction: up, down, left and right it sees wich of the 4 movements would bring the ghost closer
to the target postion(positon of player)
                    trialPosition = GetDeltaPosition(dir, rb.position);
```

```
                distances[(int)dir, i] = DistanceBetween(trialPosition, targetPosition);
            }
        }

        bestDirection = GetBestDirection(distances);
    }

    MoveInDirection(bestDirection);
}

Vector2 GetTargetPosition(GameObject target)
{
    //returns the postion of the GameObject its called with 'target'
    return target.transform.position;
}

//works out the x4 new possible postions going; up, down, left, right based on its current position
Vector2 GetDeltaPosition(Directions dir, Vector2 CurrentPosition)
{
    switch (dir)
    {
        case Directions.up:
             CurrentPosition.y += step;
            break;
        case Directions.down:
             CurrentPosition.y -= step;
            break;
        case Directions.left:
             CurrentPosition.x -= step;
            break;
        case Directions.right:
            CurrentPosition.x += step;
            break;

    }

    return CurrentPosition;
}
```

```
    //calculates the distance between x2 postions
    float DistanceBetween(Vector2 trialPosition, Vector2 targetPosition)
    {
        return Mathf.Sqrt(Mathf.Pow(trialPosition.x - targetPosition.x,2) + Mathf.Pow(trialPosition.y - targetPosition.y,2));
//the pow is sqaure btw

    }

Directions GetBestDirection(float[,] distances)
    {
        //best diresction is initialized to up
        Directions best = Directions.up;//the enum (up, down, left or right) of the bext direction
        int besti = 0;//determines which of the players is the closest

        if (Globals.Mode == Globals.Modes.normalPlay)//in this mode want the ghost to chase
        {
            //looping through both players
            for (int i = 0; i <= Globals.NumberOfPlayerAlive; i++)
            {
                //looping through all the directions
                foreach (Directions dir in Enum.GetValues(typeof(Directions)))
                {
                    //
                    if (distances[(int)dir,i] < distances[(int)best,besti])       //if there are 2 distances that are the same
it will pick the first
                    {
                        if (randomNumber.Next(1, 100) < Constants.ghostMoveProbability)
                        {
                            //only randomly updates the best direction i.e. dosent always move in the best direction
                            //giving it an oppertuniy to escape dead ends that its acught up in
                            best = dir;
                            besti = i;
                        }
                    }

                }
            }
```

```
        }

        if (Globals.Mode == Globals.Modes.powerPellet) //in this mode want the ghost to run away, otherwise its the same as
chase
        {
            for (int i = 0; i <= Globals.NumberOfPlayerAlive; i++)
            {
                foreach (Directions dir in Enum.GetValues(typeof(Directions)))
                {
                    if (distances[(int)dir,i] > distances[(int)best, besti])
                    {
                        if (randomNumber.Next(1, 100) < Constants.ghostMoveProbability)
                        {
                            best = dir;
                            besti = i;
                        }

                    }

                }
            }
        }

        return best;
    }


    void MoveInDirection(Directions bestDirection)
    {
       //changes the velocity instead of the position because want to change the movement smoothly
         switch (bestDirection)
        {
            case Directions.up:
                rb.velocity = new Vector2(0, ghostSpeed);
                break;
            case Directions.down:
```

```
                rb.velocity = new Vector2(0, -ghostSpeed);
                break;
            case Directions.left:
                rb.velocity = new Vector2(-ghostSpeed, 0);
                break;
            case Directions.right:
                rb.velocity = new Vector2(ghostSpeed, 0);
                break;

        }
    }



   //causes the ghost to return back to its 'home' position
    protected void GoHome()
    {
        transform.position = originalPosition;
    }

    //causes the ghost to change its postion to a random place in the maze, but not too close to the players home
    public void RandomRespawn()
    {
        int positionX;
        int positionY;
        bool badPosition = true;
        while (badPosition)
        {
            //ranodmly generates the x and y value of the position based on the size of the maze
            positionX = randomNumber.Next(Constants.xLeft, Constants.xRight);
            positionY = randomNumber.Next(Constants.yBottom, Constants.yTop-1);
            //dont want the ghost to randomly respwan to close to the ghosts' home, gives the ghost too much of an advantage to
kill them
            if ((positionX < -2 || positionX > 2) && (positionY < -1 || positionY > 1)) { badPosition = false;
transform.position = new Vector2(positionX, positionY); }
        }

    }
```

```
    private void OnCollisionEnter2D(Collision2D other)
    {

        switch (other.collider.tag)
        {
            //if an object with the tag of player collides with this gameObject  it increase the score and randomly espawn the
ghost to a new position
            case "player":
                switch (Globals.Mode) //need to test which mode because the action will be differnt based on if its power
pellet or not
                {
                        case Globals.Modes.powerPellet:
                        //the players gains points from eating the ghost
                        Globals.Score += eatenValue;
                        //after eaten does not respawn back to their home otherwise it can be taken advantage of
                        RandomRespawn();
                        break;
                }

                break;
        }

    }

}
```

3.2.6 Ghost Clyde

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

//this class inherits from the Ghost class (which in turn inherits from the monobehaviour class)
public class GhostClyde : Ghost
{
    System.Random randomSpeed = new System.Random();
    int count;

    // Use this for initialization
    protected override void Start()
    {
        base.Start();
        gameObject.tag = "ghostClyde"; //each ghost has individual tags to identify them
        originalPosition = new Vector2(Constants.xRight, Constants.yBottom); // clyde has his own home
        GoHome();
    }

    private void Update()
    {
        ++count;
        // changes the speed randomly every 100 frames; his 'personality' is unpredicatble speed
        if (count > 100) { ghostSpeed = randomSpeed.Next((int)Constants.speedMin, (int)Constants.speedFast); count = 0; }
    }

}
```

3.2.7 Ghost Nic

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//this class inherits from the Ghost class (which in turn inherits from the monobehaviour class)
public class GhostNic : Ghost
{

    // Use this for initialization
    protected override void Start()
    {
        base.Start();
        gameObject.tag = "ghostNic"; //each ghost has individual tags to identify them
        ghostSpeed = Constants.speedFast;
        originalPosition = new Vector2(Constants.xLeft, Constants.yTop);//Nic has here own home
        GoHome();
    }


   }
```

3.2.8 Ghost Pinky

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//this class inherits from the Ghost class (which in turn inherits from the monobehaviour class)
public class GhostPinky : Ghost {

    // Use this for initialization
    protected override void Start () {
        base.Start();
        gameObject.tag = "ghostPinky"; //each ghost has individual tags to identify them
        ghostSpeed = Constants.speedMin;
        originalPosition = new Vector2(Constants.xLeft, Constants.yBottom);//Pinky has his own home
        GoHome();
    }


   }
```

3.2.9 Globals

```
public static class Globals //static meaning that all the varibles in this class are unique
//written as a collection of functiosn for accessing each of the variables
{
    //allows me to only display certian things in the console whilst im doing testing
    private static bool dEBUG = false;
    public static bool DEBUG
    {
        get
        {
            return dEBUG;
        }
        set
        {
            dEBUG = value;
        }
    }

    // ====================== MAIN CONTROL ============================
    //relevant over persisting instances of the game

    private static int lowestHighScore, numberOfHighScores;

    private static string teamName;

    public static string TeamName
    {
        get
        {
            return teamName;
        }
        set
        {
            teamName = value;
        }
    }
```

```
public static int LowestHighScore
{
    get
    {
        return lowestHighScore;
    }
    set
    {
        lowestHighScore = value;
    }
}

public static int NumberOfHighScores
{
    get
    {
        return numberOfHighScores;
    }
    set
    {
        numberOfHighScores = value;
    }
}


// ====================== GAME FRAME ============================
//relevant each time a new game is played

private static int score = 0, levelNumber = 0, numberOfPlayerAlive;
public static int[] NumberOfLivesLeft = new int[3]; //so could index the players as 1 and 2 (and dont use 0)

public static int Score
{
    get
    {
        return score;
    }
    set
```

```
        {
            score = value;
        }
    }

    public static int LevelNumber
    {
        get
        {
            return levelNumber;
        }
        set
        {
            levelNumber = value;
        }
    }
    public static int NumberOfPlayerAlive
    {
        get
        {
            return numberOfPlayerAlive;
        }
        set
        {
            numberOfPlayerAlive = value;
        }
    }


    // ====================== LEVEL ============================
    //relevant only during each individual level

    // declared the varibale of types Modes, mode can be any one of the things from the enum declartion above, depending of the
mode diffrent actions need to be carried out, alsomost a sort of finfit state machine
    public enum Modes {normalPlay, powerPellet, endLevel, gameOver };
    private static Modes mode;
    private static int  numberOfEdiblesLeft;
```

```
  public static Modes Mode
    {
        get
        {
            return mode;
        }
        set
        {
            mode = value;
        }
    }

    public static int NumberOfEdiblesLeft
    {
        get
        {
            return numberOfEdiblesLeft;
        }
        set
        {
            numberOfEdiblesLeft = value;
        }
    }


}
```

3.2.10 HighScoreTable

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;


public class HighScoreTable : MonoBehaviour
{
    //numberOfHighScores = how many values(groups of name, score and date) will be displayed in the high score table,
numberOfEntries = how many items and or lines read in form the file
    // if there arent any values read in you want it to decide that it this high score is at the end of the empty list,
    //want such a big number so that the lowest high score that the user eneters will BE the new lowest
    int numberOfHighScores, numberOfEntries, lowestHighScore = 999999999;

    public static string[] linesOfTable = new string[]{};
    //reqire eleven for the case where there is a full list and need to push off the last item and input the users score into
the 10th position
    public static string[,] highScoreTableArry = new string[11, 3]; // with the format: name, score, date

    bool neverbefore = true;

    void Start()
    {
        //loads the file contenets at the start so that the lowestHighscore can be initialized
        ReadTableFromFile();
    }


    void Update()
    {
        //only gets called once
        if (neverbefore)
        {
            neverbefore = false;
            SortTable();
```

```
            WriteTableToFile();

        }

    }


    void ReadTableFromFile()
    {
        //tries to read the file
        try { linesOfTable = System.IO.File.ReadAllLines(@Constants.highScoreFilePath); }
        catch
        {
            //if cant find the file then sets the file to empty
            print("EXCEPTION : No file found: Highscore Table");
            //begins as if the file wa empty`
            numberOfEntries = 0;
            numberOfHighScores = 0;
            lowestHighScore = 999999999;
        }

        if (Globals.DEBUG) { print("has read the file which reads:"); }

        //VALIDATION : if the files content does not match the expected file format then will ignore, as if it was all
corrupted
        numberOfEntries = linesOfTable.Length;
        if (numberOfEntries % 3 != 0) { print("EXCEPTION: numberOfEntries is not multiple of 3 "); numberOfEntries =
0; }//detecteing if not triplets
        if (numberOfEntries > 30) { print("EXCEPTION: incorrect number of lines"); numberOfEntries = 0; }//detecting if the
file is too long  3*max instaed of 30


        //populating the 2D highScoreTableArry from the contents of the file
        for (int i = 0; i< numberOfEntries; i++) //if numberOfEntries = 0 then doesnt go into this loop at all
        {
            highScoreTableArry[(i / 3), i % 3] = linesOfTable[i]; //the second dimesion will only have three items in them
        }
```

```
        numberOfHighScores = numberOfEntries / 3; // each high score has 3 enteries


        //checking whether the score is interpratble as an intger AND saves the lowestHighScore
        for (int i = 0; i < numberOfHighScores; i++)
        {
            int INTtest;
            try { INTtest = int.Parse(highScoreTableArry[i, 1]); }
            catch { print("EXCEPTION : ReadTableFromFile : highscoretable score was NOT an int"); numberOfEntries = 0; INTtest
= 999999999; } //handles tha case where the score string isnt a number and rejects the highscores file data
            if (INTtest <= lowestHighScore)
            {
                lowestHighScore = INTtest;
            }
            else //handles the case that the values of the score are not in descending order and rejects the highscores file
data
            {
                if (Globals.DEBUG) { print("                        OUT OF ORDER " + lowestHighScore + "   " + INTtest); }
                numberOfEntries = 0;
                numberOfHighScores = 0;
                lowestHighScore = 999999999;
                break;
            }
        }

        if (Globals.DEBUG)
        {
            print("                                                                  Highscore arrayus BEFORE:");
            for (int i = 0; i < 10; i++)
            {
                print("                                                                " + highScoreTableArry[i, 0] + " " +
highScoreTableArry[i, 1] + " " + highScoreTableArry[i, 2]);
            }
        }

        //initializing the empty cells in the list highScoreTableArry to be empty strings so still displays the empty strings
instead of the string from the previous file that are no longer there
        for (int i = numberOfHighScores; i < 10; i++) { for (int j = 0; j < 3; j++) { highScoreTableArry[i, j] = ""; } }
```

```
        //saving the result, so can be used in other scenes
        Globals.LowestHighScore = lowestHighScore;
        Globals.NumberOfHighScores = numberOfHighScores;
    }

    void SortTable()
    {
        int intScore;
        bool hasItBeenAdded = false;
        if (Globals.DEBUG){ print("                                Began sort table"); }

        //inserting the users name,score,date into the list an shuffles down the scores below it
       for(int i = 0; i < numberOfHighScores; i++)
        {
            try { intScore = int.Parse(highScoreTableArry[i, 1]); } //this is double caustionary, should have already been
dealt with
            catch { print("EXCEPTION: SortTable : highscoretable score was not an int"); throw; } //handles tha case where the
score string isnt a number, throws, should never happen as have just cleaned it in readfile

            if (Globals.Score > intScore)
            {
                //have an 11th element for when have a full table, but the number of High scores is kept to 10. this also
handles the case when its not full and want to add a value onto the end
                for (int j = (numberOfHighScores); j>i ; j--)
                {
                    highScoreTableArry[j, 0] = highScoreTableArry[j-1, 0];//shift all the values down by 1
                    highScoreTableArry[j, 1] = highScoreTableArry[j-1, 1];
                    highScoreTableArry[j, 2] = highScoreTableArry[j-1, 2];

                }

                //assiging the users values into the correct position
                highScoreTableArry[i, 0] = Globals.TeamName;
                highScoreTableArry[i, 1] = (Globals.Score).ToString();
                highScoreTableArry[i, 2] = DateTime.Today.ToString("dd/MM/yy");

                // increasing the number of entries after adding the users values (unless alreday at max)
```

```
                if (numberOfHighScores < 10) { numberOfHighScores = numberOfHighScores + 1; numberOfEntries = numberOfEntries +
3; }

                hasItBeenAdded = true;
                break;
            };
        }

        //for the case when the list/file is not full AND users score has not added to the list yet: it just populates the end
of the list
        if (numberOfHighScores < 10 && hasItBeenAdded == false)
        {
            highScoreTableArry[numberOfHighScores, 0] = Globals.TeamName;
            highScoreTableArry[numberOfHighScores, 1] = (Globals.Score).ToString();
            highScoreTableArry[numberOfHighScores, 2] = DateTime.Today.ToString("dd/MM/yy");

            numberOfHighScores = numberOfHighScores + 1; numberOfEntries = numberOfEntries + 3;// increasing the number of
entries once the values have been added to the list
        }

        if (Globals.DEBUG)
        {
            print("                                                                    Highscore arrayus AFTER:");
            for (int i = 0; i < numberOfHighScores; i++)
            {
                print("                                                                " + highScoreTableArry[i, 0] + " " +
highScoreTableArry[i, 1] + " " + highScoreTableArry[i, 2]);
            }
        }

    }

    void WriteTableToFile()
    {
        if (Globals.DEBUG) { print("                                                                    should now write to
file"); }

        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@Constants.highScoreFilePath, false))
```

```
        {
            //overwriting numberOfHighScores back into the original file
            for (int i = 0; i < numberOfHighScores; i++)
            {
                for (int j = 0; j <3; j++) // still want it to be doing it in 3s beacsue the data is grouped int
name,score,date
                {
                    file.WriteLine(highScoreTableArry[i, j]); // writes each entery on a new line
                }
            }
        }

    }

    private GUIStyle guiStyleHeadings = new GUIStyle();
    private GUIStyle guiStyleScores = new GUIStyle();
    void OnGUI()
    {
        //displaying the headings
        guiStyleHeadings.fontSize = 20;
        GUI.Label(new Rect(70, 20, 100, 20), "Name: ", guiStyleHeadings);
        GUI.Label(new Rect(250, 20, 100, 20), "Score: ", guiStyleHeadings);
        GUI.Label(new Rect(350, 20, 100, 20), "Date: ", guiStyleHeadings);

        //displaying the results of the top ten highs score values
        for (int i = 0; i < 10; i++)
        {
            GUI.Label(new Rect(40, 45 + (i * 20), 100, 20), Convert.ToString(i +1) , guiStyleScores);
            GUI.Label(new Rect(70, 45 + (i*20), 100, 20), highScoreTableArry[i, 0], guiStyleScores);
            GUI.Label(new Rect(250, 45 + (i * 20), 100, 20), highScoreTableArry[i, 1], guiStyleScores);
            GUI.Label(new Rect(350, 45 + (i * 20), 100, 20), highScoreTableArry[i, 2], guiStyleScores);
        }

    }
    }
```

3.2.11 InputHighScore

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; // Required when Using UI elements.

public class InputHighScore : MonoBehaviour {

    // textbox unity object
    public InputField TheInputField;
    public string userInput;

    public void Start()
    {
        TheInputField.text = "Enter Team Name..."; // determines what is displayed in the input field before the user inputs
anything
        TheInputField.characterLimit = 20; //restricts how many characters that the user can enter
        //listener that waits the 'on end edit'event (when return is hit OR they leave the context of the InputField) then is
calls the function AsigningUserInput
        TheInputField.onEndEdit.AddListener(delegate { AsigningUserInput(); });

    }

    void AsigningUserInput()
    {
        Globals.TeamName = TheInputField.text;//storing the reslut that the usre enters into the globalS
        if (Globals.DEBUG) { print(Globals.TeamName); }
    }



   }
```

3.2.12 LoadScene

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LoadScene : MonoBehaviour {


    public void LoadSceneByIndex(int sceneIndex)
    {
        //the chosen scene's index is assigned in the unity editor window
        /*Scene Index values are:
             0 Main Control
             1 Game_frame_withLevels
             2 HighScore Display
             3 HighScore Input
         */
        SceneManager.LoadScene(sceneIndex);
    }
   }
```

3.2.13 MainControl

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MainControl : MonoBehaviour {

    //numberOfHighScores = how many values(groups of name, score and date) will be displayed in the high score table,
numberOfEntries = how many items and or lines read in form the file
    // if there arent any values read in you want it to decide that it this high score is at the end of the empty list,
    //want such a big number so that the lowest high score that the user eneters will BE the new lowest
    int numberOfHighScores, numberOfEntries, lowestHighScore = 999999999;

    public static string[] linesOfTable = new string[]{};
    //reqire eleven for the case where there is a full list and need to push off the last item and input the users score into
the 10th position
    public static string[,] highScoreTableArry = new string[11, 3]; // only top ten high scores in the table:


    // Use this for initialization
    void Start ()
    {
        //unity opertation so that the objects is retained when a new scene is loaded
        DontDestroyOnLoad(gameObject);
        ReadTableFromFile();
    }

    //loads the file contenets at the start so that the lowestHighscore can be initialized when the game is first played
    void ReadTableFromFile()
    {

        try { linesOfTable = System.IO.File.ReadAllLines(@Constants.highScoreFilePath); }
        catch
        {
            //if cant find the file then sets the file to empty
            print("EXCEPTION : No file found: Highscore Table");
            //begins as if the file wa empty`
```

```
            numberOfEntries = 0;
            numberOfHighScores = 0;
            lowestHighScore = 999999999;
        }

        if (Globals.DEBUG) { print("has read the file which reads:"); }

        //VALIDATION : if the files content does not match the expected file format then will ignore it, as if it was all
corrupted
        numberOfEntries = linesOfTable.Length;
        if (numberOfEntries % 3 != 0) { print("EXCEPTION: numberOfEntries is not multiple of 3 "); numberOfEntries =
0; }//detecteing if not triplets
        if (numberOfEntries > 30) { print("EXCEPTION: incorrect number of lines"); numberOfEntries = 0; }//detecting if the
file is too long


        //populating the 2D highScoreTableArry from the contents of the file
        for (int i = 0; i < numberOfEntries; i++)//if numberOfEntries = 0 then doesnt go into this loop at all
        {
            highScoreTableArry[(i / 3), i % 3] = linesOfTable[i];
        }

        numberOfHighScores = numberOfEntries / 3;//each high score has 3 enteries

        //checking wether the score is interpratble as an intger AND saves the lowestHighScore
        for (int i = 0; i < numberOfHighScores; i++)
        {
            int INTtest;
            try { INTtest = int.Parse(highScoreTableArry[i, 1]); }
            catch { print("EXCEPTION : ReadTableFromFile : highscoretable score was NOT an int"); numberOfEntries = 0; INTtest
= 999999999; } //handles tha case where the score string isnt a number and rejects the highscores file data
            if (INTtest <= lowestHighScore)
            {
                lowestHighScore = INTtest;
            }
            else //handles the case that the values of the score are not in descending order and rejects the highscores file
data
            {
```

```
                if (Globals.DEBUG) { print("                        OUT OF ORDER " + lowestHighScore + "   " + INTtest); }
                numberOfEntries = 0;
                numberOfHighScores = 0;
                lowestHighScore = 999999999;
                break;
            }
        }

        if (Globals.DEBUG)
        {
            print("                                                                Highscore arrayus BEFORE:");
            for (int i = 0; i < 10; i++)
            {
                print("                                                              " + highScoreTableArry[i, 0] + " " +
highScoreTableArry[i, 1] + " " + highScoreTableArry[i, 2]);
            }
        }

        //initializing the empty cells in the list highScoreTableArry to be empty strings so still displays the empty strings
instead of the string from the previous file that are no longer there
        for (int i = numberOfHighScores; i < 10; i++) { for (int j = 0; j < 3; j++) { highScoreTableArry[i, j] = ""; } }

        //saving the result, so can be used in other scenes
        Globals.LowestHighScore = lowestHighScore;
        Globals.NumberOfHighScores = numberOfHighScores;

    }
}
```

3.2.14 Player

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour {



    protected int playerID; //required to destinguish between player 1 and player 2

    public float playerSpeed;
    private Vector2 originalPosition;

    //variables to hold unity components
    private BoxCollider2D bc;
    private Rigidbody2D rb;

    private void Awake()
    {
        //attaching the componenets to the gameobjects
        bc = gameObject.AddComponent<BoxCollider2D>() as BoxCollider2D;
        bc.isTrigger = false; //when true means the objects collide
        rb = gameObject.AddComponent<Rigidbody2D>() as Rigidbody2D;
        // Dynamic behaviour causes the Rigidbody2D to react to gravity and applied forces including contacts with other
dynamic or Kinematic Rigidbody2D
        rb.bodyType = RigidbodyType2D.Dynamic;
        rb.simulated = true;
        rb.freezeRotation = true; //otherwise the gameobject will rotate when caught on corners so the keys you press no longer
correspond to the Directions you would expect

    }



    //protected allows it to be overwritten but still kept private
    protected virtual void Start ()
```

```
    {
        //want the properties of a dynamic game object but dont want it to react to gravity, so set gravity to 0
        Physics2D.gravity = new Vector2(0, 0);

        playerID = 1;
        gameObject.tag = "player";
        playerSpeed = Constants.speedFast;
        Globals.NumberOfLivesLeft[playerID] = 3;
        Globals.NumberOfPlayerAlive = 1; // index0 is player 1 and index1 is player 2
        Globals.Score = 0; //initializing the score of the team
        originalPosition = new Vector2(Constants.centre, Constants.centre); //the players' 'home'
        GoHome();//players start at home
    }

    protected virtual void FixedUpdate ()
    {
        //controls the players' movement based on the arrow keys
        transform.Translate(Input.GetAxis("Horizontal-1") * Time.deltaTime * playerSpeed, Input.GetAxis("Vertical-1") *
Time.deltaTime * playerSpeed, 0f);
    }

    public void GoHome()
    {
        // moves the player object to originalPosition
        transform.position = originalPosition;

    }

    private void DeadCheck()
    {
        //once all lives have been lost, deactivates the player gameObjects
        if (Globals.NumberOfLivesLeft[playerID] <= 0)
        {
            gameObject.SetActive(false);
            -- Globals.NumberOfPlayerAlive;
        }
    }
```

```
    //OnCollisionEnter2D :
    //Sent when an incoming collider makes contact with this object's collider.
    //Further information about the collision is reported in the Collision2D parameter passed during the call

    private void OnCollisionEnter2D(Collision2D other)//other reffres to whatever object has collided with it
    {
        //if the tag of the 'other' is one of the ghosts then it take away o alife of the player and returns them home
        switch (other.collider.tag)
        {
            case "ghost":
            case "ghostPinky":
            case "ghostClyde":
            case "ghostNic":
                switch (Globals.Mode)
                {
                    case Globals.Modes.normalPlay://if the mode was powerpellet then the player would eat the ghost not louse a
life
                        --Globals.NumberOfLivesLeft[playerID];
                        GoHome();
                        DeadCheck();
                        break;
                }

            break;//the required syntax for a case statement in c#

        }

    }
    }
```

3.2.15 Player2

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//this class inherits from the Player class (which in turn inherits from the monobehaviour class)
public class Player2 : Player {

    protected override void Start () {
        base.Start();//runs the original start on the Player object/script
        playerID = 2;
        Globals.NumberOfLivesLeft[playerID] = 3;
    }


    protected override void FixedUpdate()
    {
        //player 2 uses the keys awsd to control the movement of the player
        transform.Translate(Input.GetAxis("Horizontal-2") * Time.deltaTime * playerSpeed, Input.GetAxis("Vertical-2") *
Time.deltaTime * playerSpeed, 0f);
    }
   }
```

3.2.16 PowerPellet

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//this class inherits from the Edibles class (which in turn inherits from the monobehaviour class)
public class PowerPellet : Edible
{
    public GameObject Timer;

    public override void SetScoreValue()
    {
        scoreValue = Constants.eatenValueMedium;
    }

    public override void OnTriggerEnter2D(Collider2D other)
    {
        //completets the original OnTriggerEnter2D function inherited from edibles before adding in the extra functionality
        base.OnTriggerEnter2D(other);

        if (other.tag == "player")
        {
            Globals.Mode = Globals.Modes.powerPellet;//inistiates the power pellet mode
            Timer = GameObject.FindWithTag("timer");//begins the call for the 10 second count down
             //Generic signature for access a methods on another GameObjectc in unity:
                //      myObject.GetComponent<MyScript>().MyFunction();
                //i.e. you get the script component (Timer) from the GameObject (Timer) then call the function (SetTimer) from
the script
            Timer.GetComponent<Timer>().SetTimer(10f);
        }
    }
   }
```

3.2.17 ScoreDisplayer

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ScoreDisplayer : MonoBehaviour {

    private int count;

    //creating an object of type GUIStyle used to keep record of the porperties/characteristic of the text that is displayed
    private GUIStyle guiStyle = new GUIStyle();


    // Use this for initialization
    void Start () {
        DontDestroyOnLoad(gameObject);  // So that ScoreDisplay object is retained across Scenes
    }



    void OnGUI() //OnGUI is called every frame, so kept up to date dynamically
    {
        //displays at the bottom the game screen whilst playing
        GUI.Label(new Rect(100, 250, 100, 20), "Team Score: ");
        GUI.Label(new Rect(200, 250, 100, 20), Convert.ToString(Globals.Score));

        GUI.Label(new Rect(250, 250, 100, 20), "Lives P1:");
        GUI.Label(new Rect(315, 250, 100, 20), Convert.ToString(Globals.NumberOfLivesLeft[1]));

        GUI.Label(new Rect(250, 235, 100, 20), "Lives P2:");
        GUI.Label(new Rect(315, 235, 100, 20), Convert.ToString(Globals.NumberOfLivesLeft[2]));



        GUI.Label(new Rect(350, 250, 100, 20), "Edibles: ");
        GUI.Label(new Rect(400, 250, 100, 20), Convert.ToString(Globals.NumberOfEdiblesLeft));
```

```
        GUI.Label(new Rect(350, 235, 100, 20), "Level:  ");
        GUI.Label(new Rect(400, 235, 100, 20), Convert.ToString(Globals.LevelNumber));


        //only displays this message when its the end of the game
        if (Globals.Mode == Globals.Modes.gameOver)
        {
            guiStyle.fontSize = 40;
            GUI.Label(new Rect(100, 100, 200, 200), "GAME OVER!!!", guiStyle);
        }
        //displays this message at the end of every level
        if(Globals.Mode == Globals.Modes.endLevel) {GUI.Label(new Rect(100, 100, 300, 100), "Well done you have completed
Level:  "); GUI.Label(new Rect(320, 100, 200, 100), Convert.ToString(Globals.LevelNumber));  }

    }
}
```

3.2.18 Timer

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class Timer : MonoBehaviour {

    public float timerLength, timerLengthCherry;
    public bool coutnDown;

    public GameObject CherryGameobject;


    void Start ()
    {
        //initialises the timer
        timerLength = 0;
        coutnDown = false;
        //finds and assignings the cherry game objects
        CherryGameobject = GameObject.FindWithTag("cherry");

    }

    void Update ()
    {
        if (coutnDown == true)
        {
            //decreases the timer length by one eveery second
            timerLength -= Time.deltaTime;
            if (timerLength <= 0)
            {
                //once timer has run out; initialises the timer and mode back to normal
                timerLength = 0;
                Globals.Mode = Globals.Modes.normalPlay;
                coutnDown = false;
            }
```

```
        }

        CherryTimer();

    }

    public void SetTimer(float duration)
    {
        if (Globals.Mode == Globals.Modes.powerPellet)
        {
            //adds to the count down so that it can begin
            timerLength += duration;
            coutnDown = true;
        }
    }

    void OnGUI()
    {
        //display the timer at the bottom of the game screen with scor and lives ect...
        GUI.Label(new Rect(20, 250, 100, 20), "Timer ");
        GUI.Label(new Rect(60, 250, 100, 20), Convert.ToString((int)timerLength));
    }


    void CherryTimer() //the random respawn of the cherry is NOT depenednet on whether or not it has been eaten by the player
    {
        //adds i to the timer very sceond
        timerLengthCherry += Time.deltaTime;
        if (timerLengthCherry > 10)
        {
            //this TOGGLES the cherry active/inactive
            CherryGameobject.SetActive(!CherryGameobject.activeSelf);//if alreday active then set inactive, but if not active
then activatets it

            if (CherryGameobject.activeSelf == true) //will only call when cherry is active
            {
                //causes the cherry to randomly respawn into a new postion
                //Generic signature for access a methods on another GameObjectc in unity:
```

```
                //      myObject.GetComponent<MyScript>().MyFunction();
                //i.e. you get the script component (Cherry) from the GameObject (CherryGameobject) then call the function
(CherryPosition) from the script
                CherryGameobject.GetComponent<Cherry>().CherryPosition();

            }

            // initialises the timer
            timerLengthCherry = 0;

        }

    }

    }
```

3.2.19 WallElement

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class WallElement : MonoBehaviour {

    //variables to hold unity components
    private BoxCollider2D bc;
    private Rigidbody2D rb;

    private Vector2 whereX;

    private void Awake()
    {
        //assigning compenents to the wall elements
        bc = gameObject.AddComponent<BoxCollider2D>() as BoxCollider2D;
        bc.isTrigger = false; //when something hits no events should be triggered
        rb = gameObject.AddComponent<Rigidbody2D>() as Rigidbody2D;
        rb.bodyType = RigidbodyType2D.Static; //tops the Rigidbody2D from reacting to gravity or applied forces including
contacts with any other Rigidbody2D.
        rb.simulated = true; //does simulate unity's physics


    }

}
```

4 TESTING

Whilst I was writing my code, I created the game using an incremental method. This way I would always have a working game at each stage. For example, I started with just the player controls, and then creating the basic edibles and allowing the player to collect them. Next I introduced the ghost to chase the player as they tried to collect the edibles. (It wasn't for a while until I actually introduced a working maze.) During all these incremental stages, I performed testing to get each section to work as a robust playable game. I performed both Verification and Validation testing. (e.g. Verification testing = does the ghost calculate the correct shortest straight line distance? i.e. does it implement the design. Validation testing = does the ghost effectively chase the player in a way that is fun and challenging? i.e. does it meet the objective.). As new functional features were added I did some integration testing to be sure that the interaction with the existing parts was as expected. Sometimes these needed some adaption or tuning to work well together. This approach works well for all of the features of the game that are always in use. Some more focused testing was done for the complicated algorithms that are only run once per level, and for high scores file to check that if it is interfered with or corrupted that it wouldn't break the game.

In order to support testing and debugging I included a set of conditional print statements (controlled by a DEBUG constant so that they could be turned off when not testing). This also included some special functions for printing out the maze matrices so that the de-loop-ify algorithm could be checked.

4.1 What needs testing

What needs testing: (derived from the objectives)

- Maze algorithm
  - Creates an accurate node matrix
  - Removes all loops from the maze
  - Instantiates the maze that it has delooped?
- New maze at each new level
- Ghost chase algorithm
  - Can both chase and run away
  - Does it chase well (qualitative)
- Power pellet mode
- File reading and writing
- Score increments
- Lives lost
- Timer
- Correct movement of player in response to the keys pressed
- Completes a level

- Ends a game
- Sorts and inserts the top ten high scores
- Correct scene transitions

4.2 Testing strategy

Methods of testing I plan on using:

- Black-box testing (interfaces, do the inputs produce correct inputs):
  - Movement of player
  - Collison with walls
  - Collisions with edibles & does it increase the score/set timers
  - Collisions whit ghosts
  - When click buttons does to move to the correct scene

- White-box testing (test the possible logical paths)
  Try the paths of:
  - Play Game- get a worthy high score- enter team name-view high score table
  - Play game- don't get worthy score-return to main menu
  - View high score table-return to main menu- play game
  - View high score table-return to main menu- play game
  - - get a worthy high score- enter team name-view high score table
  - View high score table-return to main menu- play game- don't get worthy score-return to main menu

- Module Testing:
  - Test the maze algorithm
  - Test the ghost chase algorithm
  - Test the file handling

- End-User Testing: get the original two girls to play the game in real time

4.3 Test Plan and Results:

Below are a test table's which cover all the objectives and functionality of my code. Within these tests I cover the different types of test data: typical, erroneous and extreme data(when its relevant to the test) . for each test I have written a description of what the expected outcome should be then the actual results are recorded as vidoes and can be watched using this Link to the playlist of my test videos.

https://www.youtube.com/playlist?list=PLHFXFha3463XAf-oB65wmR976Wm_OBRSB

Test No.	**1**
Objectives covered	Objective No. : 1, 2
Description of Test (what)	Correct movement of players in response to the keys pressed
Purpose of test (why)	Check that both players moves correctly and independently
Expected result	a. If the up arrow or w is pressed the corresponding player should move up b. If the down arrow or s is pressed the corresponding player should move down c. If the left arrow or a is pressed the corresponding player should move left d. If the right arrow or d is pressed the corresponding player should move right
Actual Result	**Test Video 1:** 1a. 1.b 1.c 1.d https://youtu.be/gLBclHWMPrU

Test No.	**2**
Objectives covered	Objective No. : 5, 6, 7, 13,
Description of Test (what)	Score increments and lives lost
Purpose of test (why)	To check that the score increases when the different types of edibles are eaten And to check that when a player is hit by a ghost they lose a life until all lives are lost
Expected result	a. When a bit is eaten the score increments by 1 b. When a cherry is eaten the score increments by 20 c. When a power pellet is eaten the score increments by 10 d. When a ghost is eaten (in power pellet mode) the score increments by 50 e. When a ghost collides with a player (in normalPlay mode) the player loses a life and the score stays the same
Actual Result	**Test Video 2:** a. b. c. https://youtu.be/Z-Hwijt5MOE **Test Video 3:** d. e. https://youtu.be/03WjznadvH0

Test No.	**3**
Objectives covered	Objective No. : 8, 9
Description of Test (what)	Power pellet mode

Purpose of test (why)	To check that when a player collides with a power pellet gameObject the required functions of the mode are activated
Expected result	a. The ghost should flash b. The ghost should run away from the player c. The timer should have 10 seconds added to it then count down d. The player should be able to collide with the ghosts and not louse a life e. The player should be able to gain points to their score from colliding with ghost f. When the player collides with the ghosts the ghost should randomly respawn to a new position on the maze g. When the timer finishes returns to the normalPlay mode
Actual Result	**Test Video 3:** a. b. c. d. e. f. g. https://youtu.be/03WjznadvH0

Test No.	**4**
Objectives covered	Objective No. : 8
Description of Test (what)	Timers
Purpose of test (why)	To check that the timers are set to the correct value and that they count down until they are zero. Also that if the multiple power pellets are eaten that an additional 10 seconds is added onto the timer
Expected result	Power Pellet: a. When a power pellet is eaten is the 10 seconds added onto the timer b. When another power pellet mode is eaten whilst another one is still in effect, are another 10 seconds added to the timer Cherry: c. Every ten seconds regardless of whether the player eats it the cherry or not, the cherry toggles between being active and inactive
Actual Result	**Test Video 4:** a. b. c. https://youtu.be/kV7lna9wM5s

Test No.	**5**
Objectives covered	Objective No. : 3, 4 , 11, 12, 14
Description of Test (what)	Completing a level and ending a game
Purpose of test (why)	To check that when all the edibles have been eaten and not all lives lost that the game progress to a new level At each new level the difficulty increases To check that when all lives of both players have been lost that the game ends

Expected result	a. When all the edibles (bits and power pellets) have been eaten; then the game progresses to a new level, displays 'Well done you have completed Level: x' and the level number increases by one. b. Once reach level 1 ghost Pinky is introduced c. Once reach level 3 ghost Clyde is introduced d. Once reach level 5 ghost Nic is introduced e. Once reach level 7 all the ghosts move really fast f. As you go up through the levels the maze complexity/density should increase g. Once both players have lost all their lives then the 'game over' text should be dispalyed and then should leave the gameFrame scene
Actual Result	**Test Video 5:** a. b. c. https://youtu.be/NvGy91cwRBM **Test Video 6:** a. f. g. https://youtu.be/YMXG00j7cQU **Test Video 7:** d. e. f. g. https://youtu.be/N0GKH0Ou0GU

Test No.	**6**
Objectives covered	Objective No. : 12, 14,15, 17
Description of Test (what)	Correct scene transitions
Purpose of test (why)	To check that the order of the scenes is correct
Test Data	Set the variable so that a scene ends after eating only a few edibles
Expected result	a. When play game button I clicked should take you to the GameFrame scene b. When the player ends the game and is not eligible for a high score should return back to the MainControl scene c. When the player ends the game and is eligible for a high score should take the player to the enter team name scene d. Once the user has entered a team name then should to be taken to viewHighScoreTable scene e. When the user selects high score table button in the mainControl scene should be taken to the viewHighScoreTable scene
Actual Result	**Test Video 8:** a. c. d. https://youtu.be/yBd1stqNuk0 **Test Video 9:** e. a. c. d. b. https://youtu.be/VwcjSKMEjZ8

Test No.	**7**
Objectives covered	Objective No. : 10
Description of Test (what)	Ghost chase algorithm (using just a single ghost)

Purpose of test (why)	Check that the ghost can effectively chase after the players in normalPlay mode and then run away from the players when in powerPellet mode
Test data	Different scenarios to test the effectiveness of the algorithm: a. No maze b. Both the players but have one noticeably closer to the ghost than the other c. In a maze, move the player quite close to the ghost then move away d. Leave the player still in spot that is close to the ghost without any walls surrounding it e. Eaten a power pellet
Expected result	a. Without the maze the ghost should run in relatively straight line (but with a little randomness in their path) towards or away from the closest player b. The ghost should chase after/run away from the player which is closest to it c. When the player gets close the ghost should be able to follow it d. When the player is left still the ghost should be able to make its way to reach the player (as long as the maze doesn't have too many obstacles) e. Ghost should run away
Actual Result	**Test Video 10:** a. b. https://youtu.be/H4CXkP4hYT8 **Test Video 11:** c. d. e. https://youtu.be/TLxPilPH7zc

Test No.	**8**
Objectives covered	Objective No. : 11
Description of Test (what)	The different ghost personalities (all the ghosts at once)
Purpose of test (why)	To demonstrate that each of the ghosts chase the players slightly differently
Test data	Different scenarios: o No maze o With maze
Expected result	a. Ghost – chase in relatively straight line towards the players, slowly b. Pinky– chase in relatively straight line towards the players, very slowly c. Clyde – chase in relatively straight line towards the players, varying the speed randomly between slow and fast d. Nic– chase in relatively straight line towards the players, very fast
Actual Result	**Test Video 12:** a. b. c. d. https://youtu.be/jYBv35Gwako **Test Video 13:** a. b. c. d. https://youtu.be/wMBBOlzGN-U

Test No.	**9**
Objectives covered	Objective No. : 16, 17, 18
Description of Test (what)	Reading and writing from and to a file

Purpose of test (why)	Check it can handle multiple different erroneous files that would cause the program to crash if I had not dealt with them
Test Data	a. Can't find a file in the given location b. A file where its contents does not match: name, score , date format c. A file where the number of lines is not a multiple of three d. A file where the number of lines larger than 30 e. A file where the score values are not interpretable as integer f. A file where the scores are not written in descending order
Expected result	When try to open one of these erroneous files without playing a game first should delete the contents of the file and add a first entry of score 0 with the current date but no name.(Should ignore the rest of the original file even if it may have been correct) If open the erroneous file after have played a game, the first entry to the High score should be the score of the game the user just played.
Actual Result	**Test Video 14:** a. b. c. d. e. f. https://youtu.be/kxItOf-lt8Y **Test Video 15:** b https://youtu.be/mEc7-wohv0c **Test Video 16:** c https://youtu.be/mEc7-wohv0c **Test Video 17:** d https://youtu.be/LH8lQTZN_DA **Test Video 18:** e https://youtu.be/e25P1Ca8g9o **Test Video 19:** f https://youtu.be/g57RpqHPdEE

Test No.	**10**	
Objectives covered	Objective No. : 16, 18	
Description of Test (what)	Sorts and inserts the top ten high scores.	
Purpose of test (why)	To check that it can add in a different high score values into different existing tables	
Test Data	*Different table statuses* a. Empty table b. Half full table c. Table with 9 entries d. Full table (with 10 entries)	*Different score statuses* i. Not high enough to qualify into the table ii. High enough only because table is not full iii. High enough to become the new lowest high score iv. High enough to reach the top of the high score table v. High enough to reach a place in the middle of a full table
Expected result	Enter the team name, score and date into the correct position in the high score table	
Actual Result	**Test Video 20:** a. ii. https://youtu.be/kMwiIg7wr5Q **Test Video 21:** b. ii. https://youtu.be/WOCugmPp03k **Test Video 22:** c. ii. iii https://youtu.be/5-vCOHxr7h8 **Test Video 23:** d. iii. iv. v. i. https://youtu.be/0slfb5scj7M	

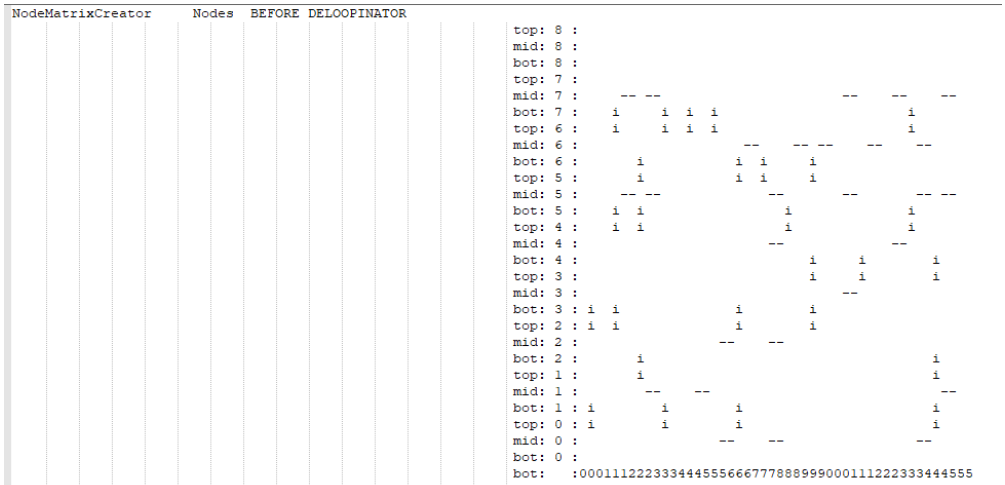
4.3.1 Testing the maze generation algorithm

Test No.	**11**
Objective covered	Objective No. : 3
Description of Test (what)	De-loop-inator
Purpose of test (why)	To check that the de-loop-inator algorithm (maze generation) gets rid of all the loops in the maze before instantiating it.
Test data	Can it de-loop a maze of randomly generated maze with low, medium and high density Can it de-loop a maze of 100% density
Expected result	Remove all the loops
Actual Result	See below:

4.3.1.1 Images of Actual result

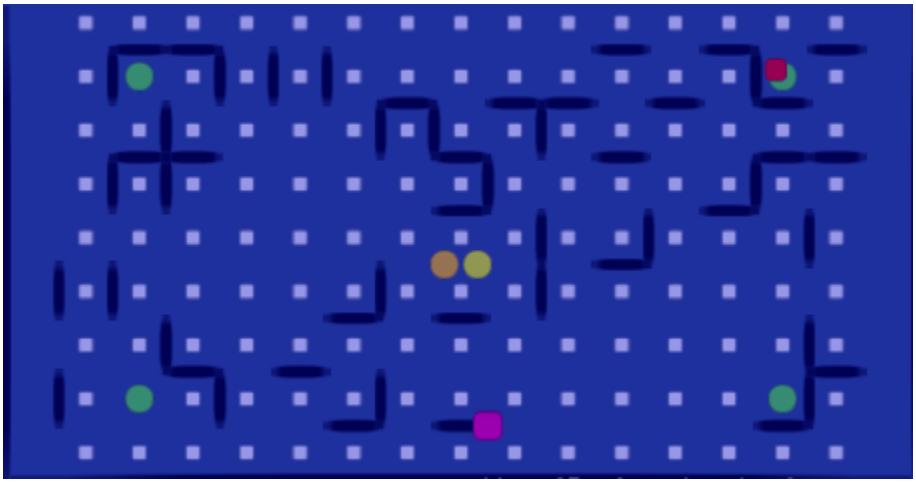
With a wall probability of 20 (will never have this actual wall probability in the game because the min is 40. But it's an example of boundary test data):

This maze did not actually have any loops to remove because it was populated at such a low density of wall elements.



```
NodeMatrixCreator     Nodes   AFTER DELOOPINATOR
top: 8 :
mid: 8 :
bot: 8 :
top: 7 :
mid: 7 :     -- --                          --    --    --
bot: 7 :    i     i  i  i                          i
top: 6 :    i     i  i  i                          i
mid: 6 :                    --    -- --    --    --
bot: 6 :       i           i  i     i
top: 5 :       i           i  i     i
mid: 5 :     -- --             --       --       -- --
bot: 5 :    i  i                 i              i
top: 4 :    i  i                 i              i
mid: 4 :                        --             --
bot: 4 :                             i     i        i
top: 3 :                             i     i        i
mid: 3 :                                   --
bot: 3 : i  i              i        i
top: 2 : i  i              i        i
mid: 2 :                  --    --
bot: 2 :       i                                    i
top: 1 :       i                                    i
mid: 1 :        --    --                             --
bot: 1 : i        i        i                         i
top: 0 : i        i        i                         i
mid: 0 :                  --    --                --
bot: 0 :
bot:   :0001112223334445556667778889990001112223334445555

NodeMatrixCreator     Walls   AFTER DELOOPINATOR
V: 7 :
H: 7 :    -- --                          --    --    --
V: 6 :   |     |  |  |                          |
H: 6 :                    --    -- --    --    --
V: 5 :      |           |  |     |
H: 5 :    -- --             --       --       -- --
V: 4 :   |  |                 |              |
H: 4 :                       --             --
V: 3 :                            |     |        |
H: 3 :                                  --
V: 2 :|  |              |        |
H: 2 :                  --    --
V: 1 :      |                                    |
H: 1 :       --    --                             --
V: 0 :|        |        |                         |
H: 0 :                  --    --                --
```

Here is the maze that ended up being displayed. See how empty and spare it is, barely a maze just a collection of some walls.
This is why I set the minimum wall density to be 40, because at this density it actually created somethings that looked mostly like a maze.



With a wall probability of 40 (initial wall probability)

```
NodeMatrixCreator    Nodes  BEFORE DELOOPINATOR
top: 8 :
mid: 8 :
bot: 8 :
top: 7 :
mid: 7 :  -- --       --            --            --
bot: 7 : i    i       i    i             i  i
top: 6 : i    i       i    i             i  i
mid: 6 :  -- -- -- --   --    -- --       -- --
bot: 6 :                i  i       i       i  i
top: 5 :                i  i       i       i  i
mid: 5 :  -- --    --       --               -- -- -- --
bot: 5 : i  i     i     i  i     i
top: 4 : i  i     i     i  i     i
mid: 4 :  --    --            --    --    -- --
bot: 4 : i     i  i  i     i     i          i     i
top: 3 : i     i  i  i     i     i          i     i
mid: 3 :  --     --    --    -- --       -- -- --    --
bot: 3 :       i     i  i  i  i          i  i
top: 2 :       i     i  i  i  i          i  i
mid: 2 :  --     --       -- --            --
bot: 2 : i  i                  i               i
top: 1 : i  i                  i               i
mid: 1 :     --    -- -- -- -- -- --       --
bot: 1 :    i     i                i
top: 0 :    i     i                i
mid: 0 :  -- -- --    -- --
bot: 0 :
bot:    :000111222333444555666777888999000111222333444555
```

```
NodeMatrixCreator    Walls  BEFORE DELOOPINATOR
V: 7 :
H: 7 : -- --       --            --            --
V: 6 :|    |       |    |             |  |
H: 6 : -- -- -- --   --    -- --       -- --
V: 5 :               |  |       |       |  |
H: 5 : -- --    --       --               -- -- -- --
V: 4 :|  |    |     |  |    |
H: 4 : --    --            --    --    -- --
V: 3 :|    |  |  |    |    |          |    |
H: 3 : --    --    --    -- --       -- -- --    --
V: 2 :      |     |  |  |  |          |  |
H: 2 : --    --       -- --            --
V: 1 :|  |                 |               |
H: 1 :    --    -- -- -- -- -- --       --
V: 0 :   |     |               |
H: 0 : -- -- --    -- --
```

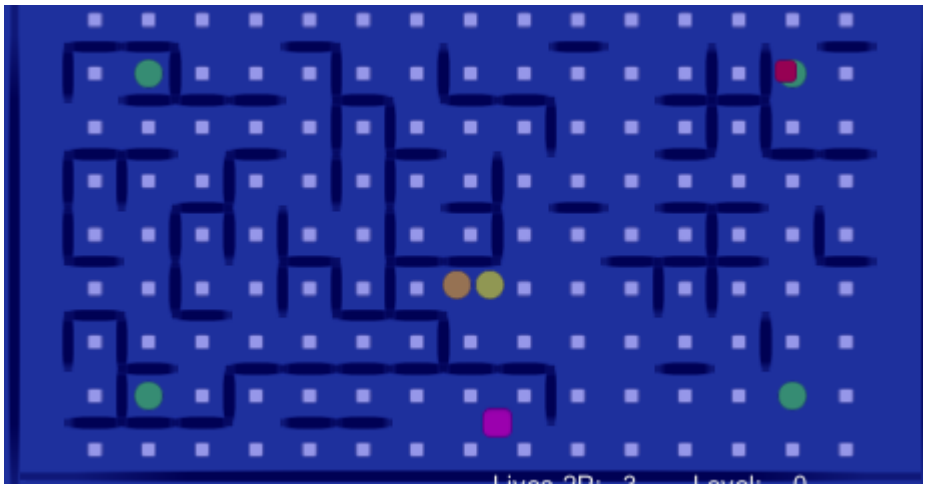
I circled all the loops in red that I expected by algorithm to deal with by removing a wall element to open up the loop.

```
NodeMatrixCreator    Nodes  BEFORE DELOOPINATOR
top: 8 :
mid: 8 :
bot: 8 :
top: 7 :
mid: 7 :  -- --       --            --            --
bot: 7 : i    i       i    i             i  i
top: 6 : i    i       i    i             i  i
mid: 6 :  -- -- -- --   --    -- --       -- --
bot: 6 :                i  i       i       i  i
top: 5 :                i  i       i       i  i
mid: 5 :  -- --    --       --               -- -- -- --
bot: 5 : i  i     i     i  i     i
top: 4 : i  i     i     i  i     i
mid: 4 :  --    --            --    --    -- --
bot: 4 : i     i  i  i     i     i          i     i
top: 3 : i     i  i  i     i     i          i     i
mid: 3 :  --     --    --    -- --       -- -- --    --
bot: 3 :       i     i  i  i  i          i  i
top: 2 :       i     i  i  i  i          i  i
mid: 2 :  --     --       -- --            --
bot: 2 : i  i                  i               i
top: 1 : i  i                  i               i
mid: 1 :     --    -- -- -- -- -- --       --
bot: 1 :    i     i                i
top: 0 :    i     i                i
mid: 0 :  -- -- --    -- --
bot: 0 :
bot:    :000111222333444555666777888999000111222333444555
```

```
NodeMatrixCreator    Walls  BEFORE DELOOPINATOR
V: 7 :
H: 7 : -- --       --            --            --
V: 6 :|    |       |    |             |  |
H: 6 : -- -- -- --   --    -- --       -- --
V: 5 :               |  |       |       |  |
H: 5 : -- --    --       --               -- -- -- --
V: 4 :|  |    |     |  |    |
H: 4 : --    --            --    --    -- --
V: 3 :|    |  |  |    |    |          |    |
H: 3 : --    --    --    -- --       -- -- --    --
V: 2 :      |     |  |  |  |          |  |
H: 2 : --    --       -- --            --
V: 1 :|  |                 |               |
H: 1 :    --    -- -- -- -- -- --       --
V: 0 :   |     |               |
H: 0 : -- -- --    -- --
```

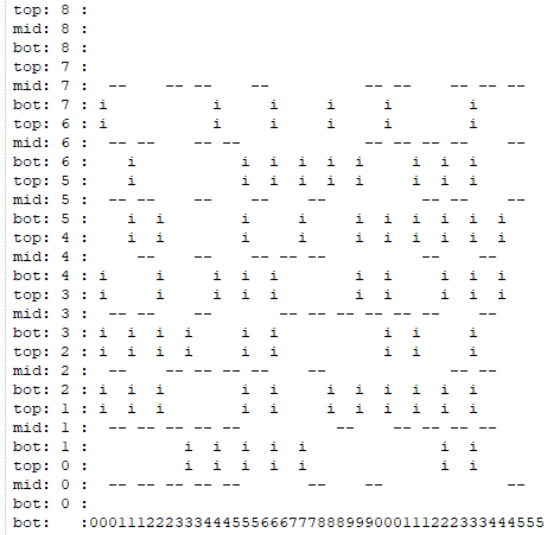
In yellow I highlighted where all of the wall elements had been deleted in order to open up the loops

The end maze that was displayed in the game, loop free.

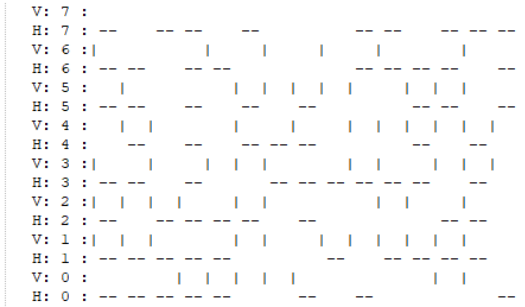


With a wall probability of 60 (test data: Typical):

```
NodeMatrixCreator     Nodes  BEFORE DELOOPINATOR
```

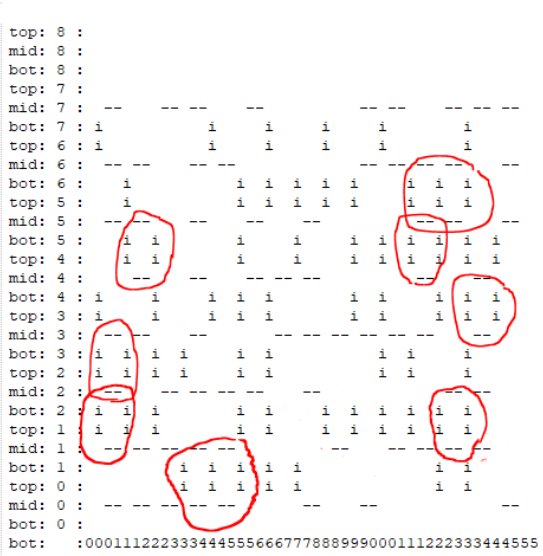



```
NodeMatrixCreator     Walls  BEFORE DELOOPINATOR
```

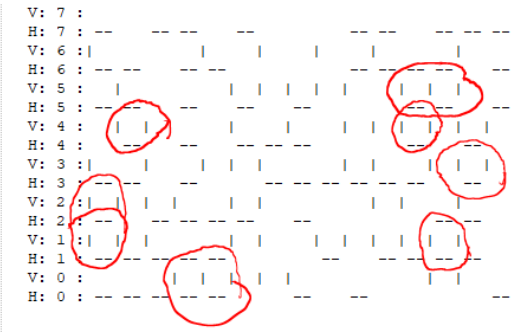


I circled all the loops in red that I expected by algorithm to deal with by removing a wall element to open up the loop.

```
NodeMatrixCreator     Nodes  BEFORE DELOOPINATOR
```



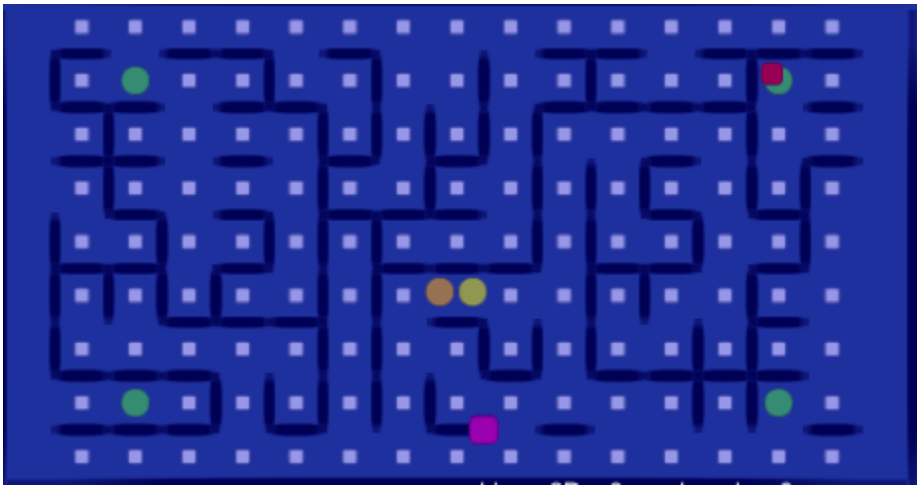
```
NodeMatrixCreator     Walls  BEFORE DELOOPINATOR
```



In yellow I highlighted where all of the wall elements had been deleted in order to open up the loops

```
NodeMatrixCreator     Nodes   AFTER DELOOPINATOR
top: 8 :
mid: 8 :
bot: 8 :
top: 7 :
mid: 7 :  --    -- --    --          -- --    -- -- --
bot: 7 : i          i     i     i     i       i
top: 6 : i          i     i     i     i       i
mid: 6 :  -- --    -- --          -- -- -- --    --
bot: 6 :    i          i  i  i  i  i          i
top: 5 :    i          i  i  i  i  i          i
mid: 5 :  -- --    --    --    --       --       --
bot: 5 :    i          i     i     i  i  i     i  i
top: 4 :    i          i     i     i  i  i     i  i
mid: 4 :     --    --    -- -- --          --    --
bot: 4 : i     i     i  i  i        i  i     i     i
top: 3 : i     i     i  i  i        i  i     i     i
mid: 3 :  -- --    --       -- -- --    -- --    --
bot: 3 : i  i  i  i     i  i           i  i     i
top: 2 : i  i  i  i     i  i           i  i     i
mid: 2 :                 -- -- --       --          --
bot: 2 : i                 i  i     i  i  i     i  i
top: 1 : i                 i  i     i  i  i     i  i
mid: 1 :  -- -- --                    --    -- -- -- --
bot: 1 :          i  i  i  i  i              i  i
top: 0 :          i  i  i  i  i              i  i
mid: 0 :  -- -- --    --       --    --           --
bot: 0 :
bot:   :0001112223333444555666777888999000111222333444555

NodeMatrixCreator     Walls   AFTER DELOOPINATOR
V: 7 :
H: 7 : --    -- --    --          -- --    -- -- --
V: 6 :|          |     |     |     |       |
H: 6 : -- --    -- --          -- -- -- --    --
V: 5 :   |          |  |  |  |  |       |
H: 5 : -- --    --    --    --       --    --
V: 4 :   |          |     |     |  |  |  |  |
H: 4 :    --    --    -- -- --          --    --
V: 3 :|     |     |  |  |        |  |     |  |
H: 3 : -- --    --       -- -- --    -- --    --
V: 2 :|  |  |  |     |  |           |  |     |
H: 2 :                -- -- --       --          --
V: 1 :|                   |  |     |  |  |     |  |
H: 1 : --    -- --                 --    -- -- -- --
V: 0 :          |  |  |  |  |              |  |
H: 0 : -- -- --    --       --    --           --
```

The end maze that was displayed in the game, loop free.



With wall probability of 100: (Data type: Extreme)

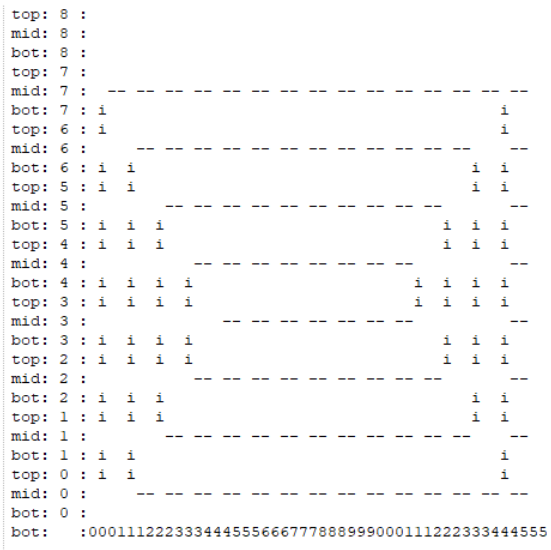
Populated all the possible horizontal and vertical wall elements

```
NodeMatrixCreator     Nodes  BEFORE DELOOPINATOR
top: 8 :
mid: 8 :
bot: 8 :
top: 7 :
mid: 7 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 7 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 6 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 6 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 6 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 5 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 5 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 5 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 4 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 4 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 4 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 3 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 3 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 3 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 2 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 2 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 2 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 1 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 1 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 1 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
top: 0 : i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
mid: 0 :  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
bot: 0 :
bot:   :0001112223334445556667778889990001112223334445555
```

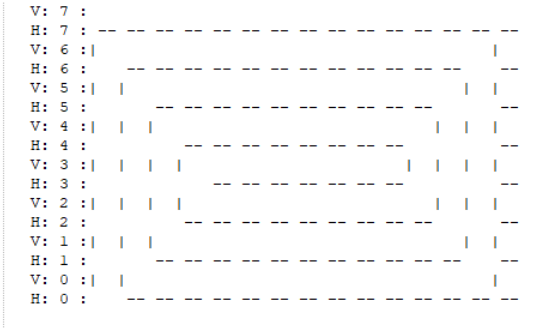
```
NodeMatrixCreator     Walls  BEFORE DELOOPINATOR
V: 7 :
H: 7 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 6 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 6 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 5 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 5 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 4 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 4 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 3 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 3 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 2 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 2 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 1 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 1 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
V: 0 :|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
H: 0 : -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

When the algorithm was applied to a maze with 100% density, it produced this interesting spiral pattern. This essentially created one enormous dead end. This meant that if you were unlucky enough to get caught in the spiral with a ghost you were doomed. Therefore, in my code I restricted the maze density so that it would not get any more dense than 70%, because of this test. If the maze gets any more dense it begins to create a spiral starting at the middle, which means that the player I too lively to get trapped and die; not fun for the player.

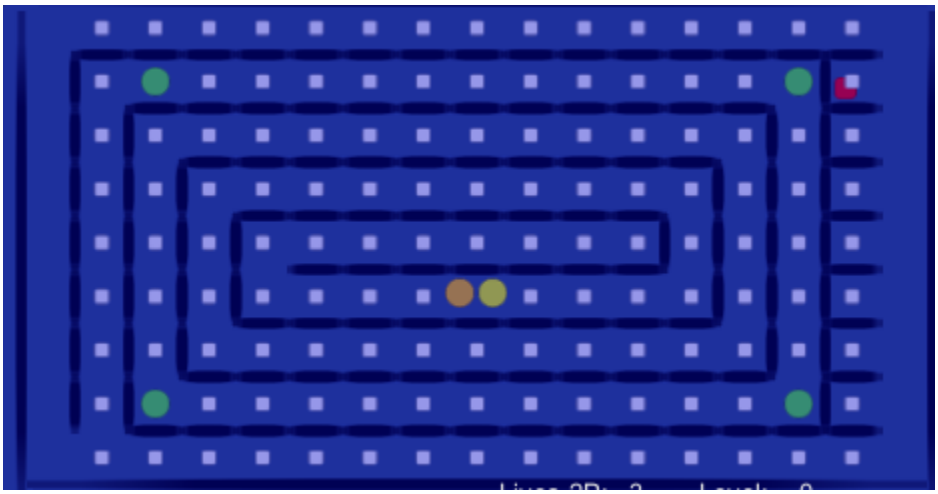
```
NodeMatrixCreator     Nodes   AFTER DELOOPINATOR
```




```
NodeMatrixCreator     Walls   AFTER DELOOPINATOR
```



This is the maze that gets displayed when you have a maze density of 100%.



5 EVALUATION

5.1 Comparing performance against the objectives

Objective No.	Objective	Perfromance Criteria	Evaluation
1	*Allow the users to play in multiplayer mode with at one other person*	Can two people play the game at the same time without disruption to the game? Need to have the same fun as a single player experience.	COMPLETELY ACHIEVED Both players can independently move their own pac-woman. Each player has their own set of keys to control the up, down, left and right movements of their player. Based in user feedback: 'more fun to be playing with someone else actually'. They enjoyed playing together towards a shared goal. Evidence in Test No.1
2	The player controls the direction in which their pac man travels (choosing between up down left or right)	*Can the player move the pac man based upon their keyboard inputs?*	COMPLETELY ACHIEVED Originally, the game was made for the arcade with a joystick controller to determine the movement of the player. However, I developed my game for a pc and chose to use keyboard keys as my input for the movements. This has not seemed to affect how enjoyably nor accessibility of the game. Evidence in test No.1 - game works as expected.
3	*At each new level the game randomly generates a maze that allows the pac-man players and ghosts to travel around without getting too stuck. All parts of the maze should be accessible by the players*	*Do the mazes vary, and are they definitely random? Also ensure that the maze is actually playable, easy enough for the players to move around in and that the players can reach all parts of the maze where there are edibles*	COMPLETELY ACHIEVED
4	*The maze generation should increase in density/difficulty as you go up through the levels*	*Does the maze generation create morecomplex mazes as the players progress through the levels? Does it get harder to complete the level?*	COMPLETELY ACHIEVED My random maze algorithm is then used at each new level to produce a new maze. The maze begins pretty 5easy but as the players progress through the levels the mazes become more dense(higher probability of the wall element being populated), making them more difficult to navigate. This is shown in Test No. 5
5	*The score count displayed during the game is live and updates regularly*	*Does the score record accurately how many points have been gained?*	COMPLETELY ACHIEVED At the bottom of the game screen the: team score, lives of each player, timer, level & number of edibles remaining are displayed. Because there are being displayed using OnGUI which is run every frame it means that their values are kept up to date every frame therefore live. The score is demonstrated to record accurately in Test No. 2

Objective No.	Objective	Perfromance Criteria	Evaluation
6	*The pac man gains points as the pac man collects: bits, ghosts and cherries*	*Does the score increase when items are collected?*	COMPLETELY ACHIEVED Whenever a player collides with an edible the corresponding score values are added to the team score. . This is shown in Test No. 2
7	*Pac man loses a life if eaten by one of the ghosts*	*Is a life lost when the pac man touches a ghosts (when not in power pellet mode)*	COMPLETELY ACHIEVED Whenever a player collides with a ghost and it is not in PowerPellet mode then the individual player's life count is decreased. This is shown in Test No 2
8	*When a power pellet is eaten (for 10 seconds) the player is be able to eat the ghost, causing the ghost to respawn. and the player gains points for eating a ghost.*	Can you eat a ghost (for 10 seconds) after eating a power pellet? Does the ghost respawn when eaten?	COMPLETELY ACHIEVED When a player collides with a power pellet PowerPellet mode is activated. This essentially means that the player now has the power to eat the ghost without losing a life. Instead of losing a life the ghost is randomly respawned to a new location and and the score increases by 50 points. Whilst in PowerPellet mode you can still eat another Power pellet, it will just increases the timer by another 10seconds so that the mode lasts longer. This is shown in Test No 3
9	*When a power pellet is eaten the ghosts run away from the player*	Do the ghosts move in a tactical format to avoid/ run away from the players when in power pellet mode?	COMPLETELY ACHIEVED As part of the ghosts chase algorithm, if the pellet PowerPellet mode is activated then the ghost chase algorithm is toggled to move the ghost in the direction which is furthest from the player. This is shown in Test No. 3, 7
10	*The ghosts chase the pac-man based on the players specific position in normal mode (no power pellet)*	Do the ghosts move in a tactical format to catch the player?	COMPLETELY ACHIEVED I wrote an algorithm which would calculate the x4 possible distance between the player and the ghost if it moved in all of the four direction(up, down, left, right). The algorithm then chooses to move the player in the direction with the shortest distance. The optimal direction would only be recalculated once between every 10-25 frames. In order to give the ghost an opportunity to travel in a certain direction for a decent amount of time before changing. I did also have to add in some randomness to whether or not the player move in the optimal direction or not. In order to help the ghost get out of dead ends. This algorithm is not always the most effective, with two players can be tactically used to keep ghosts stuck in a dead end. However in the later levels the ghost become very fast making more difficult to take advantage to that fact. (further discussed in algorithm design). The algorithm in shown in Test No. 7
11	*Certain ghosts should only get released based on which level the players are on. Each ghost should have slightly different personalities on*	Do you get more ghosts as the levels progress? Do each of the ghosts have slightly different characteristics when chasing the players?	COMPLETELY ACHIEVED Ghost (red) had a basic personality form which the other ghosts adapted from. Is the ghost that is released form the start. Pinky is almost the same as the original except moves slower. Released on the second level Clyde's speed varies randomly between slow and fast. This creates a jerky and unpredictable chasing personality. Released at level 3.

Objective No.	Objective	Perfromance Criteria	Evaluation
	how they chase the player		Nic'c speed is very fast and will run very fast to track the player down. Released at level 5. However at level 7 all the ghosts are given the the same speed as Nic in order to make it more difficult. This is shown in Test No. 8, 5
12	*The level is complete when the player collects all of the edibles*	Does the game progress to a new level once all edibles have been eaten (not including all cherries)?	COMPLETELY ACHIEVED When all the edibles have been eaten (all bits and power pellets), a short message is displayed, before moving on to the next level with a fresh maze and all the edibles re-instantiated. This is shown in Test No. 5
13	*Each player has their own set of lives.*	Do the players lose lives independently? When one player has lost all their lives does the game continue with just the remaining player?	COMPLETELY ACHIEVED The game is dependent upon both players losing all of their lives in order for the game to end. So there can be one player still playing whilst the other is dead with no lives left. Each of their lives are displayed at the bottom of the game scene. P1 = player 1 = yellow pac & P2 = player 2 = orange pac. This is shown in Test No. 2
14	*The game is over when both payers have lost all three lives*	Does the game end when all the lives have been lost?	COMPLETELY ACHIEVED Once both players have 0 lives the game ends. Depending on what score they received and the status of the high score table determines to which scene they are taken to next. This is shown in Test No. 5
15	*The game asks the players to enter a team name if their score qualifies as a high score*	Does the game ask players to enter a team name only if their score is higher than the lowest high score on the current high score table?	COMPLETELY ACHIEVED When the players' score is higher than the lowestHighScore of the current table. Then the users are taken to a scene where they can type their team name to then be placed into the high score table. This is shown in Test No. 6
16	*Saves the top 10 highest scores*	Does the game record high scores from previous games? Does this persist even when the game is shut down?	COMPLETELY ACHIEVED The highscore values are stored into a file, which persists even after the game has been shut down. So as long as the file hasn't been corrupted, then the same values from the last section of playing the game should be the same as the next time you play the game. This is shown in Test No. 9, 10
17	*Players should be able to view the high score table even if they haven't achieved a high score*	From the main menu can you view the current High score table?	COMPLETELY ACHIEVED The main menu has x2 options: play game or view the high score table. All the user has to do is click the button to take them to the scene to view the current high score table. This is shown in Test No. 6
18	*Generates a league table that is regularly updated using a sorting algorithm*	Is the high score ordered with the highest score at the top and lowest score at the bottom?	COMPLETELY ACHIEVED The code is responsible for sorting the file into the correct order and rejecting the file if it has been corrupted. The code uses a basic sort algorithm to decide where to place the players score into the table by comparing its score with the values in the table. Starting at the top until it finds a score that it is higher than, then inserts it. This is shown in Test No. 10

5.2 Getting and analyzing independent feedback

Once I had completed the game, I got the two original girls to have a go playing the finished product. Overall, the feedback was very positive and they both enjoyed the game. They also agreed that the game met all of the objectives/original requirements. Their only reservations were maybe that at the lower levels were too easy. The maze was almost a little too big and 135 edibles was a lot of edibles to have to collect just to complete the first level. This slightly detracted from the game because it meant that it required a little patience to get into the more exciting and challenging parts of the game.

5.3 How could the outcomes be improved

If I had had more time to spend on my game, here are a few improvements I would consider adding.

Possible improvements:

- More sophisticated ghost chase algorithm (there are pros and cons – the ghosts might be too good and that would change how the game was played)
- The graphics, make it look pretty
    - Add aminations for the moving sprites
    - Add intro animations between the levels – e.g. creating the maze
    - Make the text and font fonst have a common more aesthetic theme
- Add encryption to the high score table file so that it cannot be tampered with.
- Make the maze smaller, too big takes too long and gets boring (for impatient player)- or even make the maze grow as the levels get harder, but start with a smaller maze
- Add an instructions/intro to teach players how to play/what the different edibles meant who the ghosts were
- Make it so you can have single player mode as well as multiplayer mode
- Make it so you could play against the other player, instead of in a team together with them
- Make it so that the player can choose the difficulty they want to start at. Experienced players don't need to go through the easier early levels every time.

5.4 Final Conclusion

Overall, I feel that my project was successful because I managed to create a working game that is enjoyable. The project was also successful because it met all of the requirements that was set at the start in my analysis. I will now try to share this game with more people. Hopefully this will help more girls to become engaged with computer games and computer science.