Project for Gesture Based UI

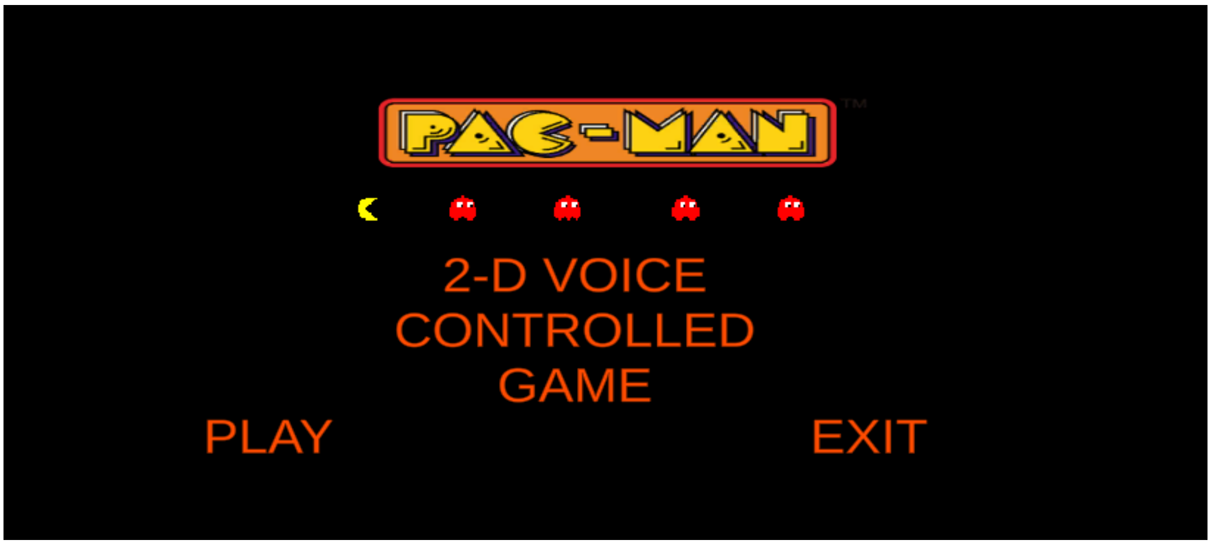
Development Documentation

Student Name: Steven Joyce

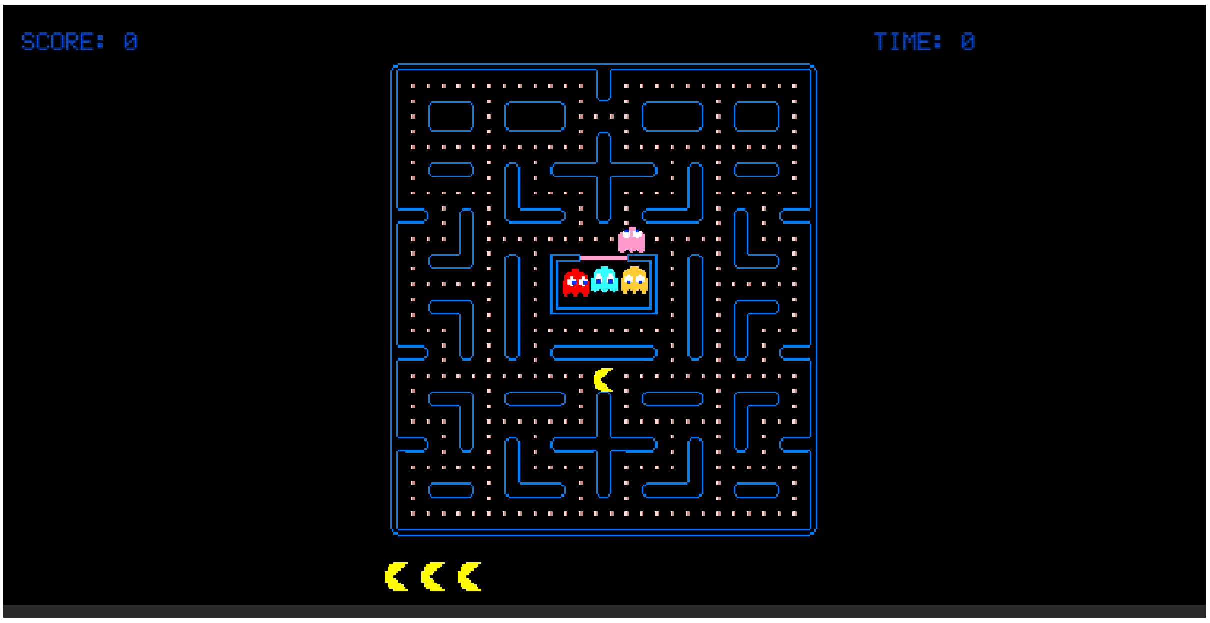
Student ID: G00362012

Purpose of the application

The purpose of this project was to create a 2-dimensional version of Pacman with a twist. It is a survival game, the objective of the game is to survive for 2 minutes without being killed by the ghost.



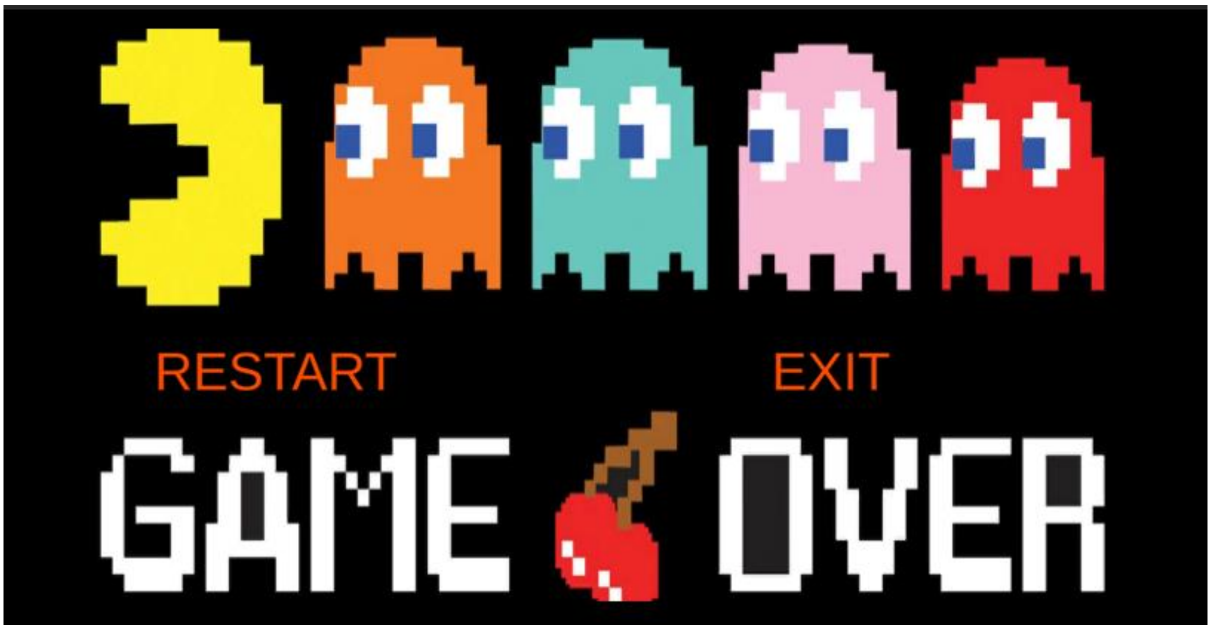
The game starts with a menu that allows the user to launch the game or exit the application. The menu only has text due to it only being controlled with voice gestures such as play or exit.



When the user starts the game from the menu they will be brought to this scene. This scene is the actual game that this project was created to develop. The yellow object found in the maze is Pacman. Pacman is what the player has control over and it can only be controlled by the user's voice. Pacman is able to move left, right, up and down or stop by using the gestures in the application and will collect pacdots along the way which add to the score in the top left hand corner of the screen.

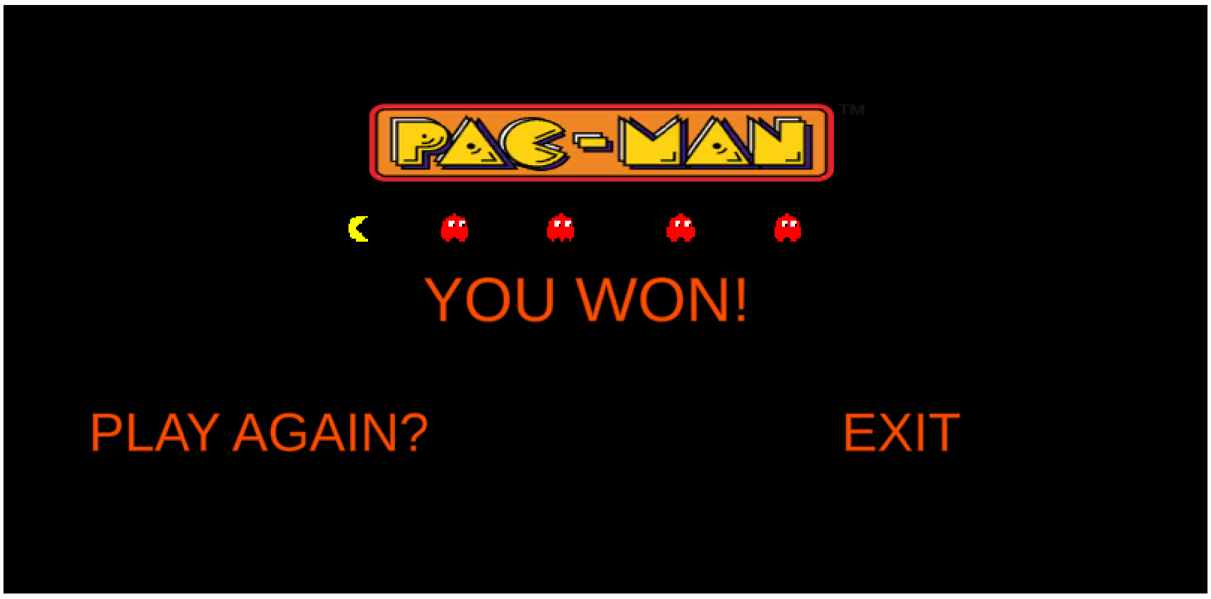
The 4 game objects in the middle of the maze are the ghosts. The ghosts are the enemies of Pacman in the game and they will be controlled by waypoints that set a path for the ghost to always follow. This gives the user an obstacle to overcome, can they dodge the ghosts for 2 minutes without being hit 3 times and dying.

The life count of Pacman can be located in the bottom left hand side of the maze, the count starts at 3 and will decrease whenever the ghost hit Pacman. If the life count reaches 0, the game over screen will be launched – you lost.



The game over scene is set up like the menu scene due to it only being controlled by the user's voice. You can either restart the game or exit the application depending on what gesture you activate.

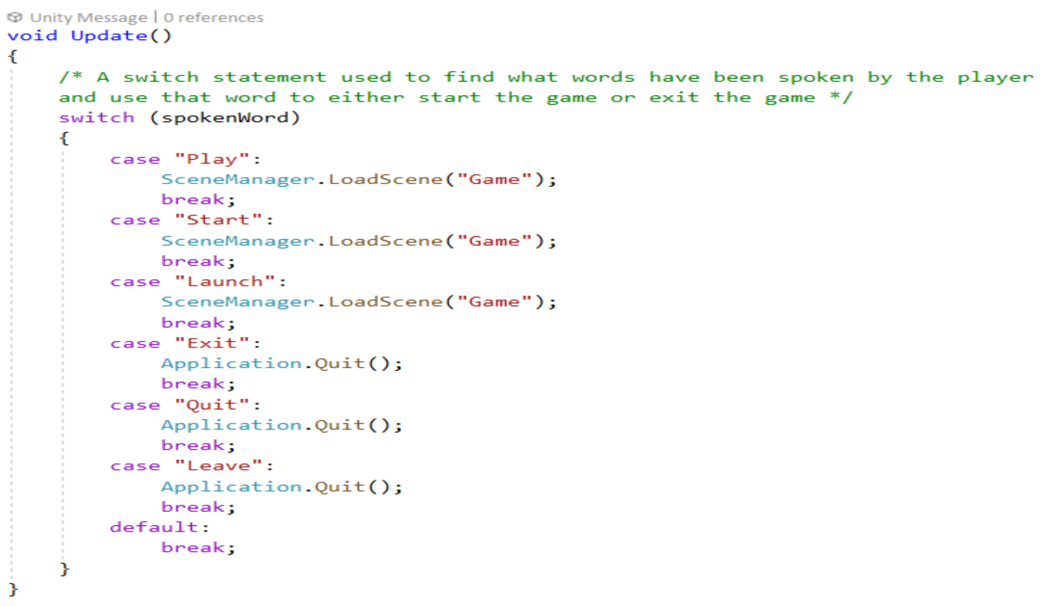
The timer is located on the top right hand corner of the Game screen, it will start at 2 minutes and go down to 0. When it reaches 0, the player wins. The player will then be sent to the win scene.



The win scene is also only controlled by gestures activated by voice recognition. The player can either restart the game or exit the application depending on what gesture they have launched.

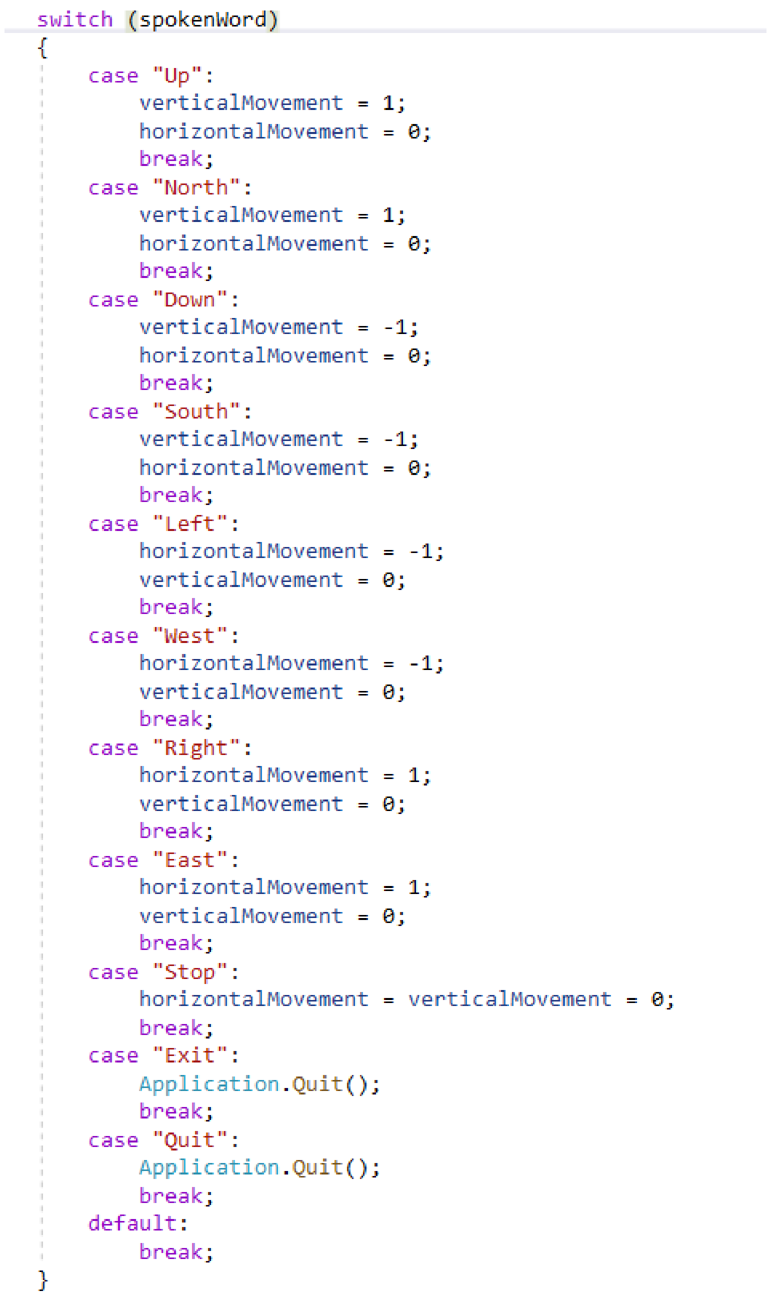
Gestures identified as appropriate for this application and Architecture for the solution

The type of gestures utilized in this game is voice control. The voice control is the only input that will allow any function of the game to be activated. Without the ability to input words into your computer device this game will only stay on the main menu screen indefinitely.



```
Unity Message | 0 references
void Update()
{
    /* A switch statement used to find what words have been spoken by the player
    and use that word to either start the game or exit the game */
    switch (spokenWord)
    {
        case "Play":
            SceneManager.LoadScene("Game");
            break;
        case "Start":
            SceneManager.LoadScene("Game");
            break;
        case "Launch":
            SceneManager.LoadScene("Game");
            break;
        case "Exit":
            Application.Quit();
            break;
        case "Quit":
            Application.Quit();
            break;
        case "Leave":
            Application.Quit();
            break;
        default:
            break;
    }
}
```

The game begins with the menu scene. This scene is only controlled by the user's voice. To play the game the user can speak the following words: Play, Start or Launch. To exit the application the user can speak: Exit, Quit or Leave. The are read in to spoken word before being passed into the above switch statement and the correct action performed.



```
switch (spokenWord)
{
    case "Up":
        verticalMovement = 1;
        horizontalMovement = 0;
        break;
    case "North":
        verticalMovement = 1;
        horizontalMovement = 0;
        break;
    case "Down":
        verticalMovement = -1;
        horizontalMovement = 0;
        break;
    case "South":
        verticalMovement = -1;
        horizontalMovement = 0;
        break;
    case "Left":
        horizontalMovement = -1;
        verticalMovement = 0;
        break;
    case "West":
        horizontalMovement = -1;
        verticalMovement = 0;
        break;
    case "Right":
        horizontalMovement = 1;
        verticalMovement = 0;
        break;
    case "East":
        horizontalMovement = 1;
        verticalMovement = 0;
        break;
    case "Stop":
        horizontalMovement = verticalMovement = 0;
        break;
    case "Exit":
        Application.Quit();
        break;
    case "Quit":
        Application.Quit();
        break;
    default:
        break;
}
```

When the user goes to the game screen, they will be able to control Pacman be saying on of these words Left, Right, Up, Down, North, South, East, West, Stop, Exit or Quit .

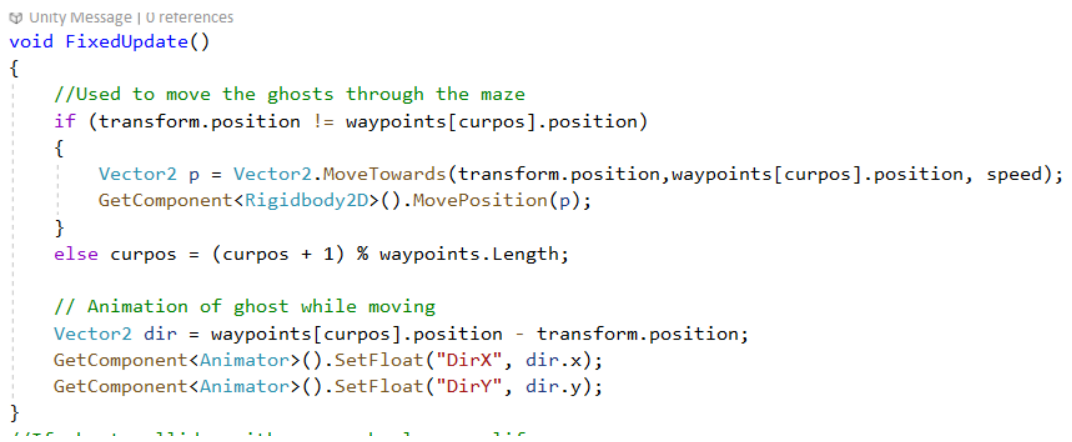
- To move Pacman left - the phrases are Left or West
- To move Pacman Right – the phrases are Right or East
- To move Pacman Up – the phrases are Up or North
- To move Pacman Down – the phrases are Down or South
- To stop Pacman from moving- the phrase is Stop
- To exit the game – the phrases are Exit or Quit

The above image illustrates how the words are recognized to perform the actions required for running the game. The vertical and Horizontal movements are then used in the following line of code to move Pacman's position on the game scene.

```
rb.velocity = new Vector2(horizontalMovement * speed, verticalMovement * speed);
```

I have set the speed of Pacman's movement to 5, I found this allowed the gestures to react at a quick rate without making the flow of the game too slow. I first tried 10 but the speed was too quick and the reaction was too slow to respond to the gesture command while 1 and 3 was too slow that Pacman could easily be caught by the ghosts.

To test the ghost speed I first set it to 0.5 but this was too fast for the user to be able to navigate the game with voice controls. 0.1 was a small bit too slow so I ended up setting the speed at 0.2. I found the game flows better with it set at this value.



```
Unity Message | 0 references
void FixedUpdate()
{
    //Used to move the ghosts through the maze
    if (transform.position != waypoints[curpos].position)
    {
        Vector2 p = Vector2.MoveTowards(transform.position,waypoints[curpos].position, speed);
        GetComponent<Rigidbody2D>().MovePosition(p);
    }
    else curpos = (curpos + 1) % waypoints.Length;

    // Animation of ghost while moving
    Vector2 dir = waypoints[curpos].position - transform.position;
    GetComponent<Animator>().SetFloat("DirX", dir.x);
    GetComponent<Animator>().SetFloat("DirY", dir.y);
}
```

The ghost Ai code uses the waypoints given to each ghost to set a path for the ghost to follow. The ghost animation that is linked to that movement is activated by giving the animator the x and y value of the ghost position. The ghost is then moved at the speed set in the game and in the direction of the next waypoint. The ghost moves along the path in a constant loop until the game ends. There is no stoppage in the ghost's movement when the last waypoint is reached, it just starts the path again.

Hardware used in creating the application

I wanted to build a game utilizing a Kinect but the Kinect I ordered was missing the connection part and my replacement piece took too long to arrive, leaving me to have to create a voice controlled game in Unity instead.

The only hardware needed to run this application is a microphone that can connect to a PC. The microphone is used to input the words that run the game.

Conclusions & Recommendations

I learned how to use a game object to change another game objects animation state. I learned how to create a path for a game object through a maze while not colliding with the maze walls. I could get the Pacman to move via voice control through the maze correctly and it makes the game look unfinished

I honestly did not enjoy creating this project, I find voice controls with unity not reliable and can cause the user to not be able to move the player every time a word that is inside the dictionary of the game script is called. It leads to frustration with the game and tools that run it. I wish I could have used Kinect to build my game but the part arrived too late for it to be viable.