CS 4610/ CS 7610

**Final Project Report**

**Spring 2020**

**Group 6**

Nick Henry, Sarah McLaughlin, Jay Toebben, Tiffany Young

****

* A comprehensive description of the system you developed.
* Explain your methods, some illustration figures of the results, as well as any remaining issues and potential solutions.
* upload all the source code, executable, report onto the Canvas.
* Team member list with role assignments for each member.
* Bring a hardcopy to the class.
* Include a Youtube link of your project

Description:

Team Members and Roles:

Nick Henry

* Pacman
  + Controls
  + Animations
  + Animator
  + Character Transitions
* Added Collision
  + Collision to maze, and pacman
* Level Resetting
  + Clearing a level of pacdots resets all of the components to the beginning position in order to start a new level

Sarah McLaughlin

* Pacdot/Big Pacdot
  + Placements and functionality
* Start page and Game Over page
  + Scene navigation
* Audio and Audio volume controls
  + Options > Sound bar
  + Audio source volume player preference
* Scoreboard and High Score
  + High Score player preferences
* Levels
  + Level display
  + Level update to ghost speed
* Lives
  + Pacman.jpg
  + Add/Delete with life changes

Edward Toebben

* Maze Creation in blender
* Nav Mesh
* Ghost AI
* Pacman control
* Teleporters
* Assisted other team members in troubleshooting

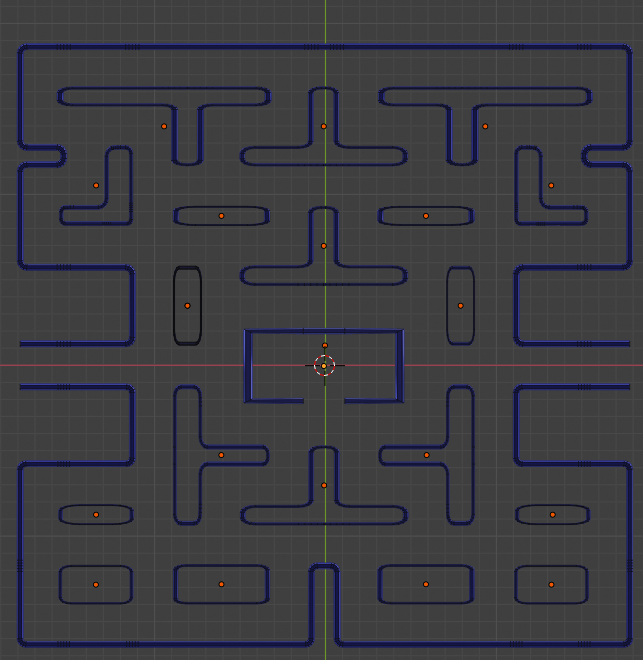
Tiffany Young

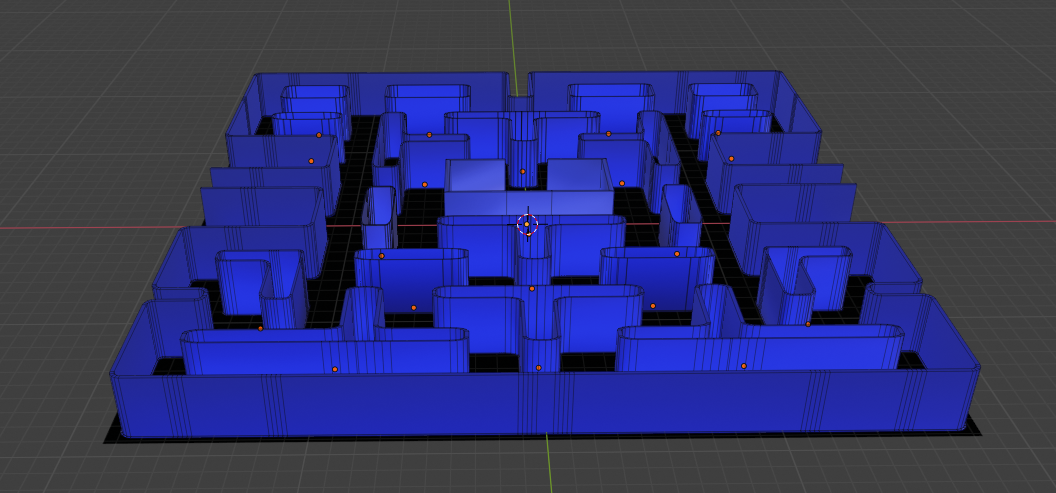
* Ghosts
  + Creation in blender 2.82a
  + Ghost Modes “Scared” and “Normal”
    - Animations and Animations Timer for both modes
    - Scared mode-- running away from PacMan
    - Scared mode-- collision with Pacman (Getting eaten)
* Fruit
  + Created in blender 2.82a
  + Fruit Spawn Script

**YouTube Link to Demo:** [**https://www.youtube.com/watch?v=HwACHf1d5WM&list=TLPQMzAwNDIwMjDDM0SSyQNqmg&index=1**](https://www.youtube.com/watch?v=HwACHf1d5WM&list=TLPQMzAwNDIwMjDDM0SSyQNqmg&index=1)

**Maze**

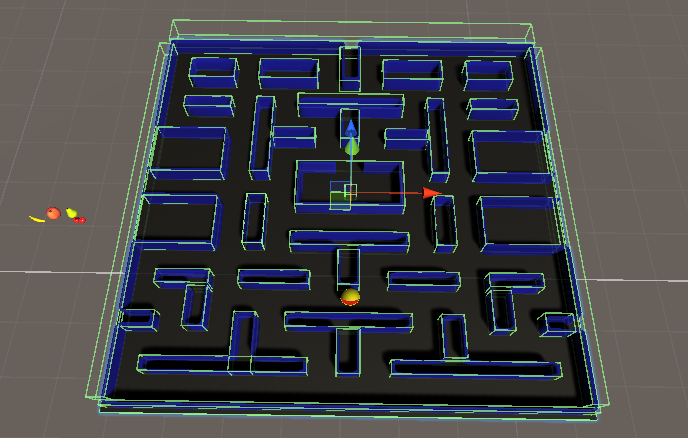
The maze like the ghosts was made using Blender 2.82a. We used an image of the pacman game to map the outline of the maze. After the maze was made we tested out various heights to decide which one gave us the right look we wanted. Once the height was decide on, we added the floor and set it as a prefab so the team could work on it without having variance in the beginning.



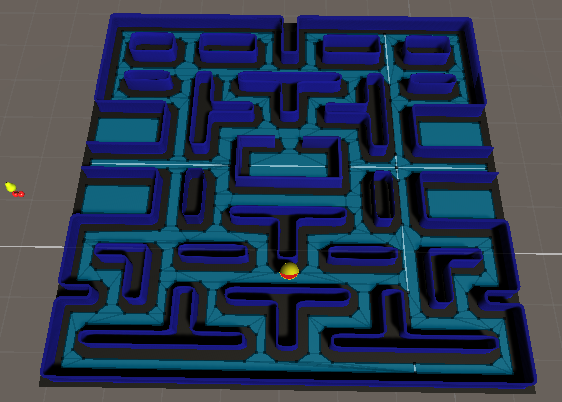


Nav Mesh

After creating a maze our team started working on how to keep the Ghost and Pacman on the path and not clip into walls. The first idea was to create collision blocks, in Unity, around every wall and have the items “bounce” off the walls. This ended up failing as characters would not collide twice in a row.

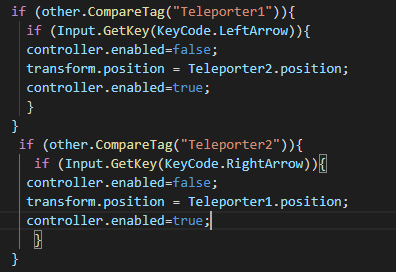


After some research we discovered a feature of Unity called Nav Mesh. Nav Mesh creates virtual walkways for agents. As you can see in the picture below the maze has a specific area the characters can travel. The nav mesh is essential for pacman controller and the ghost AI.



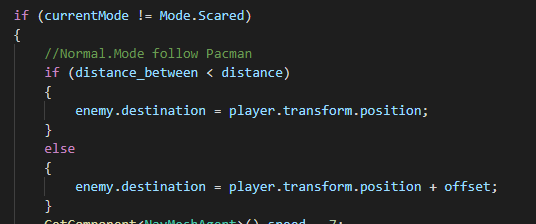
**Teleporting** (done in PacmanController.cs)

The final thing to add to the maze was transporters in pacman, the pacman would need to be teleported from one side of the map smoothly and instantly. To do this the team set up box colliders that when hit would give the object a new position. This issue with this is the team was using a controller for pacman and would result in the new position being ignored. To combat this we would turn off the controller change position and then turn the controller back on. Now that pacman appeared on the opposite side another problem arose, pacman would hit the collider box on the side we just travelled to and bounce back and forth endlessly. The team resolved this by adding a conditional direction statement.



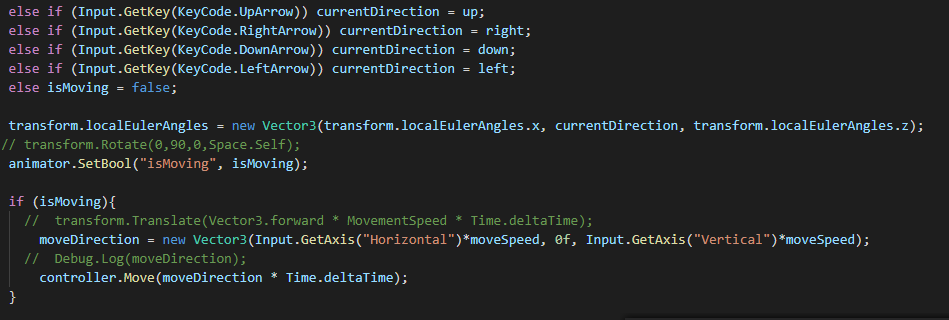
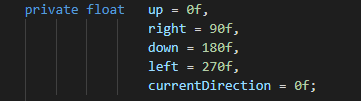
**Ghost AI**

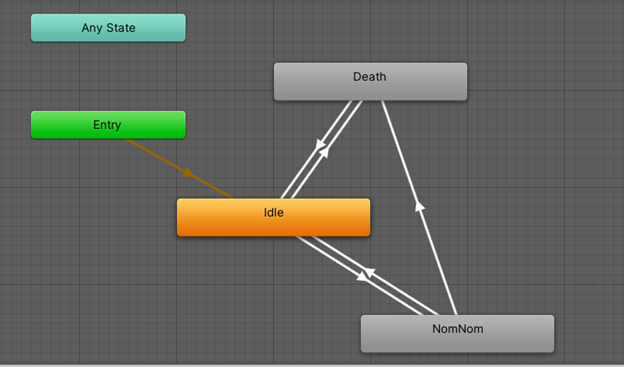
The ghost AI started just following pacman using a NavMeshAgent, which would find pacman and map the shortest path to him. The issue with this is all four ghosts end up walking the same path and are easily avoided. To combat this we add an offset to pacman allowing the ghost to surround him, and once the ghost is within a certain range or distance the ghost will be given a direct path to pacman. In this code each ghost is given its own offset creating a set of ghosts that are harder to avoid.



**Pacman Controls** (done in PacmanController.cs)

When controlling pacman our team wanted pacman’s controls to act much like the classic 2D game. To move pacman was not that difficult, however we also wanted pacman to turn as he was led down different paths. To move pacman we used the built in unity controller and mapped the direction keys to a Vector3 and fed it to the controller. To get pacman to rotate we set angular values and transformed them to using the EulerAngles method.

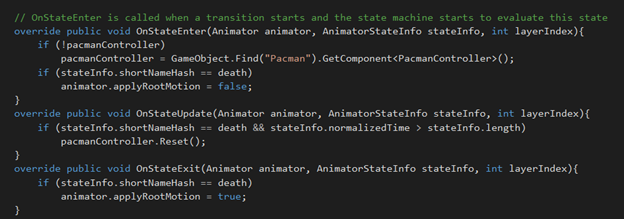


**Pacman Animator**

In order to get pacman to do certain actions, an animator was made to monitor the different states that pacman has. Whenever Pacman is not moving, he stays in the idle state. NomNom is the state that he is moving, and Death is when he touches a ghost. The switches between the states are boolean variables that are used inside of the PacmanController.

**Pacman DeathState**

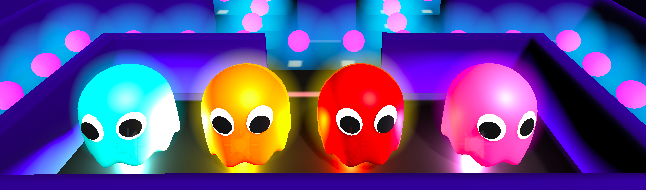
To achieve the death animation, and resetting of Pacman’s position, we needed to add a script to the death state.



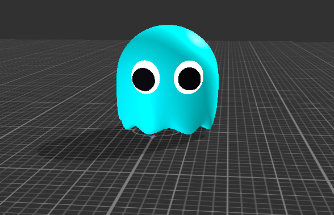
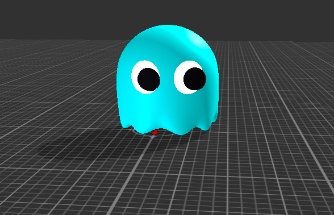
**Ghosts**

The ghosts Pinky, Blinky, Inky, and Clyde were made using Blender 2.82a and given their color and animations in Unity. The material that creates their skin color uses a standard Shader with the Opaque rendering mode. Emission is also used which creates a very vibrant glowing color. To achieve their glow the “Halo” effect and point lights were used. The ghosts have two modes: “Normal Mode” and “Scared Mode”.

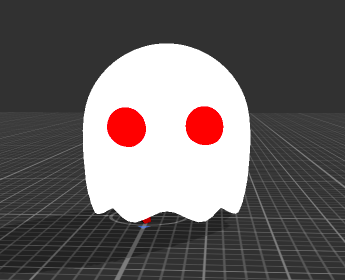
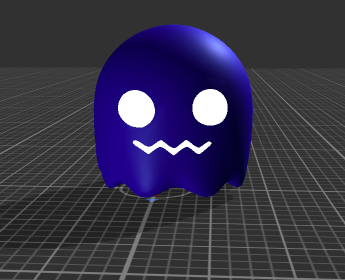
**Normal Mode Ghosts: Inky, Clyde, Blinky, Pinky**



When in Normal Mode the eyes and legs of the ghosts move side to side. This is achieved using the “NormalGhost<*ghostname>*” animation.

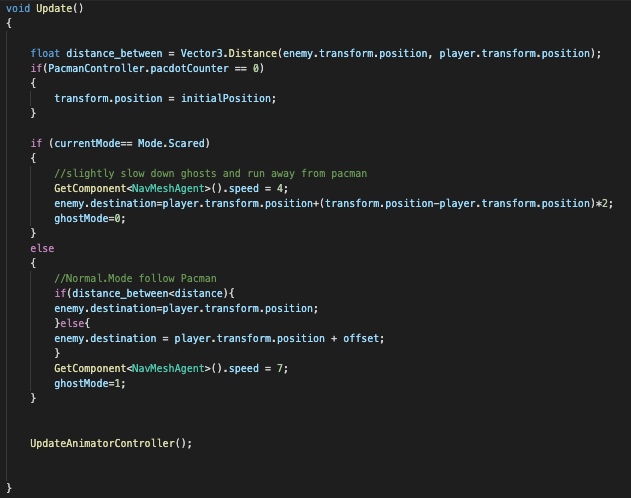


When in Scared mode the ghosts turn blue and their eyes, mouth, and “legs” move side to side. Scared mode lasts 10 seconds and at the 7th second the ghost will blink White.



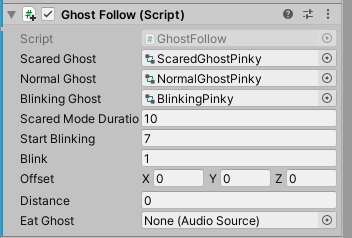
**“GhostFollow.cs” Script**

When the ghosts are in Normal Mode they track down Pacman and when they are in Scared Mode they run away from Pacman. The **currentMode** variable is used to determine which mode the ghosts are in and which direction they should head in. As seen below, when the currentMode is set to  **Mode.Scared**, the ghosts are directed to a point that is away from Pacman based on the current distance between pacman and the ghost. When **currentMode** is set to **Mode.Normal** the ghosts track down pacman based on its current location. An offset variable was used to prevent ghosts from being too “aware” of where pacman is.

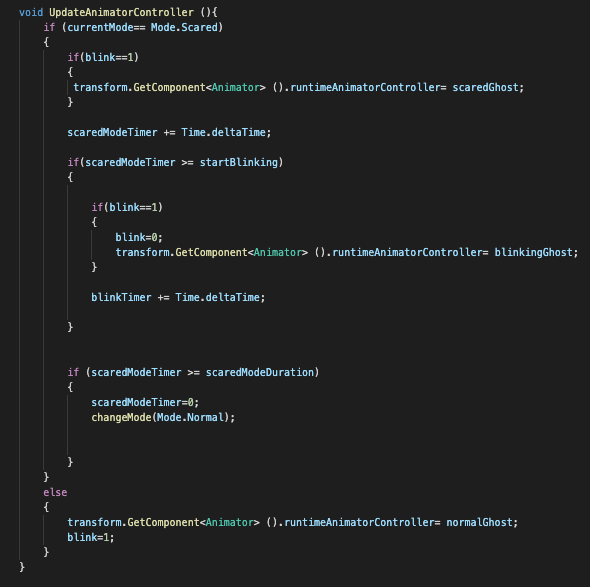


The **UpdateAnimatorController()** function is used to update the animation of the ghost.

When ghosts are in Normal Mode, the animation is set to an Animator Controller that is set in Unity. Normal Mode does not need a timer.



However, when ghosts are in Scared Mode a timer is used. Scared mode lasts 10 seconds. At 7 seconds the ghosts start to blink white to warn the user that scared mode is almost over. As seen below if **currentMode** = **Mode.Scared** and **blink** == 1 then the animator controller is set to “scaredGhost” **scaredModeTimer** will get incremented and the rest of the if conditions will not be met. Once the **scaredModeTimer** is greater than or equal to the **startBlinking** variable (which is set at 7) **blink** will be set to zero and the ghosts will be set to the “blinkingGhost” animations. The **blinkTimer** will be incremented and now because **blink** == 0, the ghost will not be set back to the “scaredGhost” animation for the remaining 3 seconds of the Scared mode duration.



When ghosts are in Normal Mode they can kill Pacman (set in the PacmanController script) and when they are in Scared Mode they can be eaten by Pacman. This is achieved using the **OnTriggerEnter** function. When **ghostMode** is set to zero (0 means that the ghost is in scared mode and 1 means the ghost is in scared mode) and a ghost comes into contact with Pacman, NavMeshAgent is disabled on the ghost and the ghost is set back to its initial position. The **initialPosition** variable was set in the start() function at the beginning of the program.



The NavMeshAgent is disabled so that the ghosts stop moving when being placed at their initial position. Once they are placed, it is enabled again so that they can start over their movements as normal ghosts. Ghosts are worth 100 points when they are eaten so the Score.score variable is incremented by 100. Pacman’s health is also incremented in this function and the eat ghost audio is played.

**REMAINING ISSUES WITH GHOSTS:**

When ghosts become scared, a public static variable called “ghostMode” is toggled to allow pacman to eat the ghosts rather than to be killed by the ghosts. Because this variable gets toggled at different times by each individual ghost,the code breaks and pacman dies when trying to eat other scared ghosts. This does not happen each time, but there are times where there is a random glitch and this occurs. A possible fix for this bug is to remove the “enemy” tag and make a tag for each ghost and treat them individually with their own toggle variables.

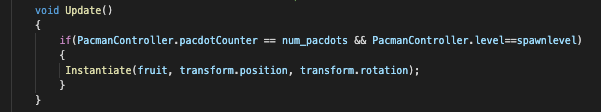
**Fruits**

****

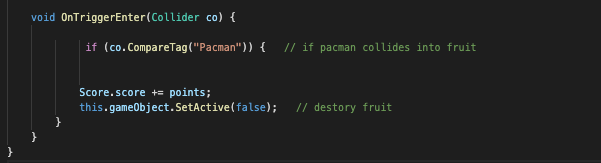
Fruits were created using blender 2.82a and given their color and texture images in unity. Fruit will spawn during specific levels using the “FruitSpawn.cs” script. Each fruit is worth a specific amount of points. Details can be seen in the table below:

|  |  |  |
| --- | --- | --- |
| **Fruit** | **Points** | **Level** |
| Cherries | 100 | 1 |
| Pear | 150 | 2 |
| Banana | 200 | 3 |
| Apple | 300 | 5 |

**“FruitSpawn.cs” Script**



Empty objects with the naming convention “<fruitname>Spawn<#>” (example “CherrySpawn1”) were created for each spawned fruit and placed at the desired spawning location. This object has the sphere collider component added to it for collisions. When the number of Pacdots in the maze equals the variable num\_pacdots and the game level equals the variable level,then an instance of the fruit will be created. Once pacman collides with the fruit the instance and the spawn location is removed from the maze. If a fruit was not eaten during the level in which it spawns and the level has incremented, the fruit will be removed from the maze.



**Score System**

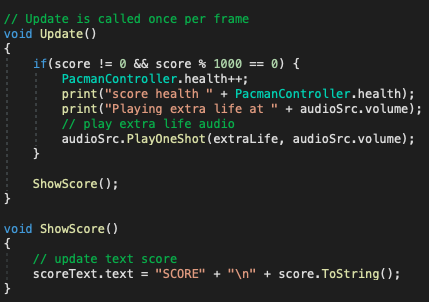
**Points System**

Some of our points are based off of the original points system:

* Pacdot = 10 points
* Big Pacdot = 20 points
* Eating ghost in “Scared Mode” = 100 points
* Eating fruit = 200 points (see table above)

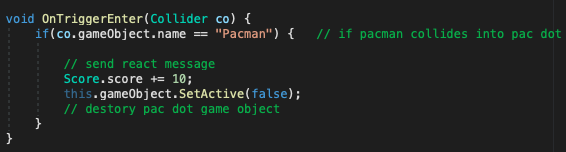
The score is displayed dynamically on a TextMesh Pro scoreboard that is changed on multiple OnTriggerEnter() functions when Pacman collides into them.

Score.cs is called to update the TextMesh Pro on the Update() function so that each collision can be accurately displayed.



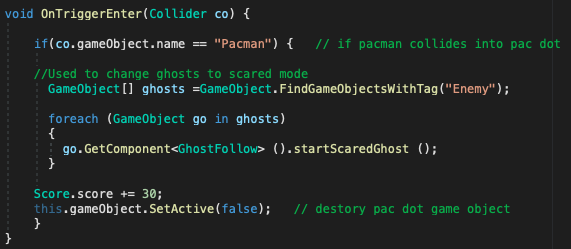
* **Pacdot.cs**

Pacman collision with a Pacdot adds 10 points to the Score() variable score.

****

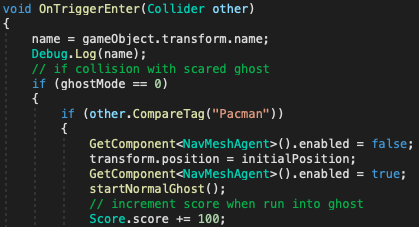
* **Bigpacdot.cs**

Pacman collision with Big Pacdot adds 30 points to the Score() variable score.

****

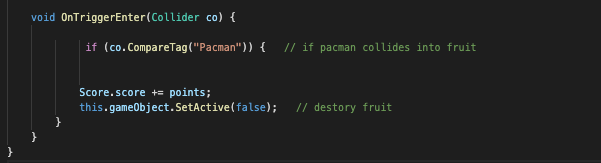
* **GhostFollow.cs**

Pacman collision with a ghost in “Scared Mode” adds 100 points to the Score() variable score.

****

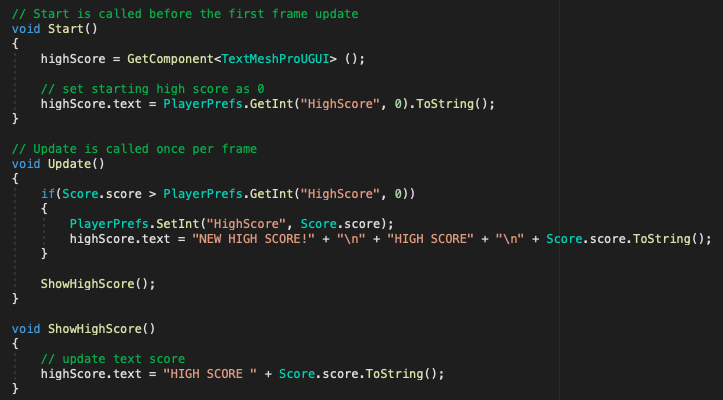
* **FruitSpawn.cs**

Any of the fruit spawns adds 200 points to the Score() variable score.



**High Score** (done in HighScore.cs)

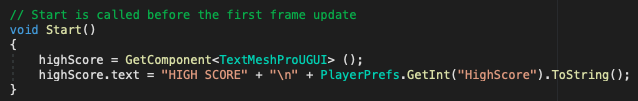
The High Score is stored in player preferences and is displayed dynamically in Update() and only changed when the original High Score is beat. Initially, if the game has not yet been played on the system before, the score is saved as 0. After that point, and point that is greater than the previous High Score is saved to the player preferences as the ending score.



With the High Score stored in user preferences, it is possible then to also display the player preferences high score on multiple scenes, including the Main scene:



For the other two scenes, Game Over and Start Page, it is easy to display the high score by getting the player preferences for the High Score value using PlayerPrefs.GetInt()

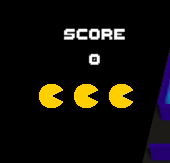


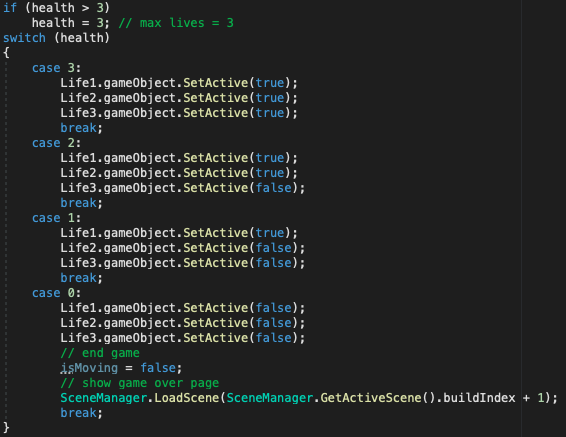




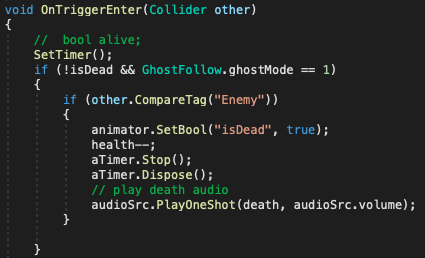
**Health System** (done in PacmanController.cs)

The Health System is initialized with three lives, displayed by .jpg images of type Sprite, where depending on collisions with enemies or helpful objects the lives can be incremented or decremented. The correct number of lives are displayed.



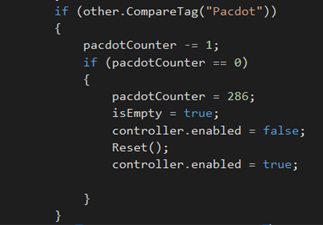
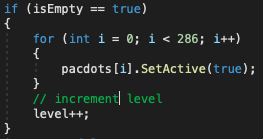


Initially, there was an issue in decrementing the lives as Pacman often runs into more than one ghost and lives were decremented multiple times on collision, rather than just one life. Using a timer that is set to a similar amount of time it takes for Pacman to respawn, the life is only decremented once before respawning when colliding with ghost(s) in “Normal Mode”.



**Level System** (done in LevelUp.cs and PacmanController.cs)

In order for a level up to happen, Pacman has to eat all of the Pacdots on the stage. This is achieved with a counter that decrements every time Pacman collides with a Pacdot and eats it. Once the counter hits 0, all of the Pacdots will repopulate and Pacman will be reset to his spawn location.



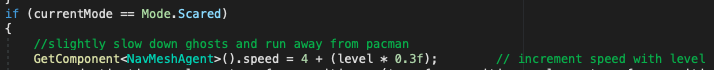
When the level is incremented in PacmanController.cs, LevelUp.cs compares the old level value to the new level value and displays a level up and the level number TextMesh Pro.

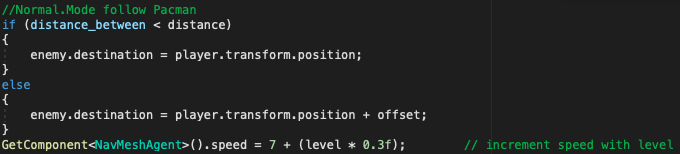




**Level Controls** (done in GhostFollow.cs)

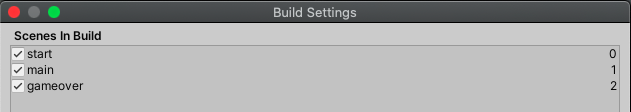
The level controls the speed of the ghosts when in both “Normal Mode” and “Scared Mode”.





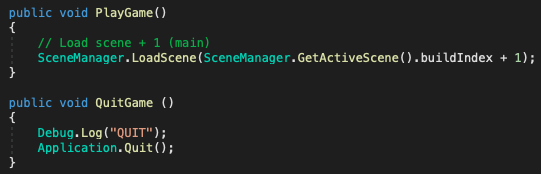
**Scene Control** (done in Unity Build Settings)

Our three scenes are included in Unity Build Settings and are displayed accordingly depending on the action and current scene displayed.



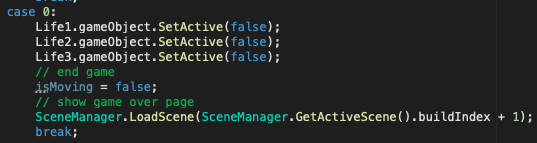
* **MainMenu.cs**

On selection of Play within the main menu, PlayGame() is called which loads the scene after the Start Scene, which is Main.

****

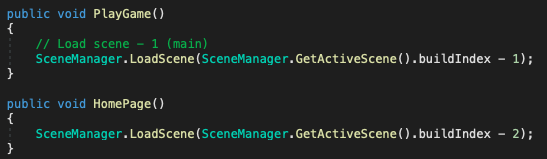
* **PacmanController.cs**

With zero lives left, the game ends and loads the scene after the Main Scene, which is end game.

****

* **GameOver.cs**

Game over has two buttons, either Play or Home, each assigned to the PlayGame() and HomePage() function accordingly. Play loads the scene before it, Main. Home loads 2 scenes before it, the Start page.

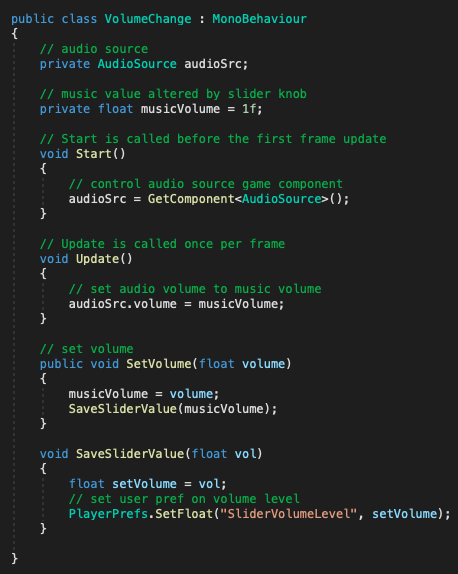


**Audio Controls** (done in VolumeChange.cs)

Audio volume is controlled within the Main Menu > Options on the Start page. The volume is saved into the player preferences by using the UI slider.

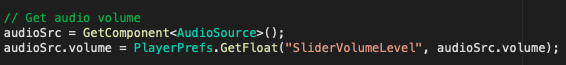


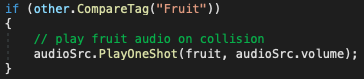
Similarly to the High Score functionality, the Audio Volume is saved within the player preferences so that it can be referenced in other scenes. The audio is initially set to full volume (1f), and as the slider is adjusted, the level is saved as the player preference audio volume.



For each item that plays any audio source, the audio source volume is retrieved from player preferences using PlayerPrefs.GetFloat() to set the PlayOnOneShot volume.

PlayerController.cs example of audio:





An issue I ran into was attaching multiple audio sources to one game object and calling those. I originally had used the Play() function and found in Unity Docs that creating Audio Clips and calling PlayOneShot() would call the particular audio source to be played at the saved Audio Source volume level.

