**Lab Week 4**

*50.033 Game Design and Development*

1004147 --- Riley Riemann Chin

**Are you participating in the Weekly Lab competition?** ~~Yes~~/No

**Provide the YouTube/other platform link to your screen recording:**

<https://youtu.be/Qs2hay9C54g>

**Provide the link to your lab repository:**

<https://github.com/rileychin/50.033-Game-Dev-Labs/tree/lab4>

**Describe what you have done to achieve the desired checkoff requirement for this lab:**

[Your **high level** description here]

* You don't need to be too specific. This is just to assist us when we check your repository
* Scripts added, Assets added if any
* General modifications that you have done: eg animating the enemies, implementing FSM for the NPCs, etc.
* Use the ObjectPooler and spawn 2 enemies at a time. The Player can “Kill” the enemy any way you want, such as stepping from the top. This will increase score and spawn one new enemy. Make sure you show that you use an object pooler by recording the Hierarchy as well.
  + To spawn 2 enemies at a time, I just edited SpawnManager to spawn 2 greenEnemy when the script starts. This will call the spawn from pooler object twice for the green enemy and just add it into pooledObjects. The pooledObjects are already instantiated with the objects by setting the settings from the gameObject settings directly.
  + Graphical user interface, application

    Description automatically generated
  + I will expand pool for both so that it will add into the existing pool if there is not enough objects in pool
  + Text

    Description automatically generated
  + startSpawn is disabled so we don’t spawn any goomba’s at first, but the gomba will be added into the existing pool because of the objectpooler settings for expandPool
  + Text

    Description automatically generated
  + Here’s the hierarchy for enemyspawnpool when the game starts
* Create some “interactive” bricks to collect coins or whatever rewards that will increase score but spawn one new enemy. It is up to you to determine how many max enemies can be present at a time. In the gif below, we limit total number of gomba enemies and green enemies to be 5.
  + We first create a coin prefab, attached with a coin.cs script.
  + This script will literally only have the OnTriggerEnter2D() function for when the player (haven’t coded out that portion) will collide with the coin, this will cause the cascading function calls for CentralManager.centralManagerInstance.increaseScore()
  + I also added a collectCoin() function in CentralManager which adds the spawnNewEnemy method in SpawnManager, such that it will also spawn a new enemy.
  + Text

    Description automatically generated
  + To limit the number of enemies I just used an if condition in ObjectPooler.cs which will not spawn and add a new object to pool if the count is already >= 5.
  + Text

    Description automatically generated
  + It is a simple solution, but the problem is that if it reaches the maximum 5 pooledobjects, then it will return null and not spawn a new object. But it will just spawn until the maximum object count and then stop adding items to pool from there.
* Player will “die” under certain conditions (up to you), and this will cause the enemy to “rejoice”. Both actions of player dying and enemy rejoicing must be clear to the player. In the gif sample below, player will die if it collides with the enemy from anywhere except from the top. The enemies have this cute little dance when the player dies, and player’s death is animated as well.
  + Originally when mario dies in lab 2 I had to disable time.timescale to be 0.0f, which effectively pauses a game and stops all movement. However in this lab I changed it such that I stop mario and the enemies from moving by vector2(0,0) for both. They will then call the marioDeathDance() coroutine and enemydance() coroutine where it is just effectively flipping the X transform of the gameobject periodically.
  + Text

    Description automatically generated
  + Here is an example of the enemy dancing.
  + Text

    Description automatically generated
  + Calling EnemyRejoice.
  + Text

    Description automatically generated
  + Calling PlayerDiesSequence()
  + All these scripts will be called because of the event defined in GameManager, this will help by making it easier to compile all the things that will happen when the player dies. For every method added to the event, we have to remove it when the script destroys, if not it will continuously add it when the scene is reloaded again!
  + A screenshot of a computer

    Description automatically generated with medium confidence
  + Adding the last line at the start() method
  + Text

    Description automatically generated with low confidence
  + Removing it when the scene restarts.
* Use AudioMixer in your project, in any way you want.
  + By basically following the tutorial I added the audioMixers for the different sound effects for certain events.
  + A screenshot of a computer

    Description automatically generated with medium confidence
  + Graphical user interface

    Description automatically generated
  + The audio mixer is then attached to the output defined by the audio mixer created.
  + Additional audio mixers are added for the background and the brick breaking sound and also the jump sound. (will not be shown)
* Powerups: create two different powerups from anywhere e.g: the question boxes. Player can “collect” it as shown, and “cast” it later. Bind the “casting” to two distinct keys, signifying that the player consumes it. The effect should disappear after a fixed number of seconds. For example, the red mushroom allows the player to jump higher for 5 seconds. The max number of powerups that can be collected can be fixed to 1 for each type for simplicity.
  + Also done by following the tutorial, first started by defining the scripts needed to upgrade the player speed or jump.
  + Then creating a powerupmanager script that will manage the object getting consumed by the player, showing up on the UI, etc.
  + Then the last step is to just cast the to consume the powerups.
* Usage of ScriptableObject to store some unchanged constants like enemy speed, fixed patrol locations, etc. This won’t be visible easily in the game, so you need to record yourself clicking to the scriptable object instance and show that the values are used by other scripts.
  + ScriptableObjects acts kinda like a database that stores many constants.
  + Text

    Description automatically generated
  + Here are the list of constants provided.
  + Text

    Description automatically generated
  + Example in debris.cs
  + Text

    Description automatically generated
  + Example in breakbrick.cs
  + Text

    Description automatically generated
  + Example in enemyController.cs