Challenge 5 - GameManager

The GameManager and Singleton

Many times when we develop game, we have some generic variables and functions that does not really belongs to any GameObjects (player, enemy, wall and etc). Variables like isGameOver, highscore, timer, number of enemies left or functions like saving highscore or restart game.

Therefore, we will have a GameManager GameObject with a GameManager.cs script to hold all the generic variables and functions.

Understanding that we will only have one GameManager in a game, we will also use this software design pattern called Singleton where it restricts the instantiation of class to one “single” instance. This is useful when you need only 1 object to coordinate actions across the game.

At the end you will be able to:
- Create a GameManager GameObject and Script
- Create 2 different prefab “AddEnergy” and “MinusEnergy”
- Create a timer in GameManager

Completed sample: https://simmer.io/@itegpd/~90a6d840-0331-c9ac-08f5-96b9b3418ec8

Part 1 – Creating the GameManager and its script

1. Create a new Unity project and name it as “00_HanselKoh_GameManager”
2. Rename the SampleScene as GameScene
3. Create an empty GameObject and name it as GameManager
4. Create a GameManager script inside GameManager GameObject.
5. Inside GameManager script add the following class variable
   ```
   public static GameManager instance;
   ```
6. Inside Start() of GameManager script add the following
   ```
   if (instance == null)
   {
       instance = this;
   }
   ```
   The above check if the instance variable is null (empty) before assign `this` (which refer to the GameManager class itself as the instance variable.

Part 2 – Creating the Prefabs

1. Create 2 Cube with different material colors (Green and Red), and name them as “AddEnergy” and “MinusEnergy” respectively
2. Create a folder named as “Materials” in the Project panel and put both Green and Red material inside.

3. Create a folder named as “Prefabs” in the Project panel and drag both AddEnergy and MinusEnergy from Hierarchy panel into the Prefab folder.

   Prefabs is short for prefabrication (Premade objects), where we can duplicate many of them later.
4. Delete both AddEnergy and MinusEnergy that is in the Hierarchy panel
5. That is how to make a GameObject into a prefab.

Part 3 – Using GameManager to Instantiate (spawn) prefabs

1. Inside GameManager script, create 2 references to take in both AddEnergy and MinusEnergy prefabs.
   ```
   public GameObject addEnergyPrefab;
   public GameObject minusEnergyPrefab;
   ```
2. Drag both prefabs from the Prefabs folder to the GameManager component on the Inspector panel.
3. Inside GameManager script, create another int variable to define the number of prefabs to Instantiate (spawn).
   ```
   public int numberOfSpwan;
   ```
4. Inside the Start() of GameManager, add the following to spawn the prefabs.
   ```
   // Instantiate addEnergyPrefab at random position //
   for (int i = 0; i < numberOfSpwan; i++)
   {
       Vector3 randomPos = new Vector3(Random.Range(-15, 15),
                                       0,
                                       Random.Range(-15, 15));

       Instantiate(addEnergyPrefab, randomPos, Quaternion.identity);
   }
   ```

   In the above code, the amount that we are going to instantiate (spawn) will depend on the variable *numberOfSpawn*.

   Then randomise the position using *Random.Range(MinValue, MaxValue)* in the X and Z axis.
5. Remember to set the value 10, in the inspector for *numberOfSpawn* variable.
6. You should be able to see the AddEnergyPrefab instantiate over an area of -15 to 15 range.

Part 4 – Instantiate (spawn) random prefabs

1. Now that we can instantiate random position, we are going to random between AddEnergyPrefab, and MinusEnergyPrefab.
2. Replace the existing Instantiate() with the following
   ```
   if (Random.Range(0, 2) < 1)
   {
       Instantiate(addEnergyPrefab, randomPos, Quaternion.identity);
   }
   else
   {
       Instantiate(minusEnergyPrefab, randomPos, Quaternion.identity);
   }
   ```
   What we are doing above, is to random value from 0 to 1, and if random number is < 1, we will spawn addEnergyPrefab, else spawn minusEnergyPrefab.

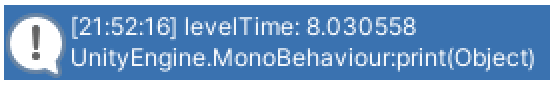
3. Test to see if you can spawn both types of prefabs.

Part 5 – Making a timer

As we have used Time.deltaTime many times for movement, we can also use deltaTime to keep track of how much time has passed.

A timer in a game can be counting (1, 2, 3, 4, 5) or down (10, 9, 8, 7, etc).

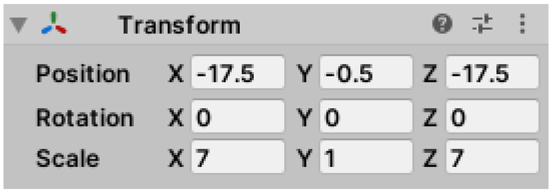
For this lab, we shall create a timer that counts down.

1. Inside GameManager script, create a public variable
   ```
   public float levelTime;
   ```
2. Set levelTime variable to 10 from the inspector panel
3. Inside the Update() of GameManager, we will start decreasing levelTime with Time.deltaTime.
   ```
   void Update()
   {
       // check if there is still time left //
       if (levelTime > 0)
       {
           levelTime -= Time.deltaTime;
           print("levelTime: " + levelTime);
       }
       else
       {
           levelTime = 0;
           print("Times up!");
       }
   }
   ```
4. Observe the timer decrease in the Console panel
   
   ```
   [21:52:16] levelTime: 8.030558
   UnityEngine.MonoBehaviour:print(Object)
   ```
5. We shall better improve it with format like minutes:seconds:milliseconds
6. Add the following function in GameManager Script
   ```
   public string FormatTime(float time)
   {
       int minutes = (int)time / 60;
       int seconds = (int)time - 60 * minutes;
       int milliseconds = (int)(1000 * (time - minutes * 60 - seconds));
       return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds, milliseconds);
   }
   ```
7. Replace the print() with the use of FormatTime() to format it in to min:sec:msec format.
   ```
   levelTime -= Time.deltaTime;
   //print("levelTime: " + levelTime);
   print("levelTime: " + FormatTime(levelTime));
   ```

With the above done, we have now got our GameManager! We shall make some game.

<u>Setting up the game play and play area</u>

Import all the given assets
- Character3.unitypackage
- Environment.unitypackage
- Materials.unitypackage
- Props.unitypackage
- QuestionMark.unitypackage
- Target.unitypackage

Floor GameObject
- SM_PolygonPrototype_Buildings_Floor_5x5_01P
- 

Transform

Position	X	-17.5	Y	-0.5	Z	-17.5
Rotation	X	0	Y	0	Z	0
Scale	X	7	Y	1	Z	7

Character GameObject
- Cube GameObject
- Create a Character script
- Movement
  - A or left arrow to face left, and move forward direction
  - D or right arrow to face right, and move forward direction
  - W or up arrow to face front, and move forward direction
  - S or down arrow to face back, and move forward direction
  - Speed 10
  - Timebased
- Implement a variable EnergyCount for Character script
- Collision
  - Collision with AddEnergyPrefab will + 5 to energyCount
  - Collision with MinusEnergyPrefab will -5 to energyCount

Camera
- Create a camera script
- Follow the player

That is all for the basic game play.

Enhancing the game

1. Create UI Text to display the timer
2. Create UI Text to display the character's energy
3. Stop character movement when timer is 0
    a. Use GameManager.instance.levelTimer to check for time.
4. Everytime character collected an AddEnergyPrefab, use GameManager to add time.
    a. You may create a new function in GameManager, name it AddTime()
    b. Use GameManager.instance to call AddTime() to add time.
5. Everytime character collected an AddEnergyPrefab, use GameManager to spawn **4 more prefabs** (AddEnergyPrefab or MinusEnergyPrefab) at random position
    a. You may use AddTime() to instantiate more prefab.
6. Everytime character collected an MinueEnergyPrefab, minus 25 energy of energyCount
7. Create a lose scene when energyCount is < 0, create a function in GameManager to change scene to "LoseScene"
8. When levelTime < 0, change scene to lose scene
9. Create a win scene when energyCount is > 50, create a function in GameManager to change scene to "WinScene"
10. Inside WinScene, create a WinSceneController script, and implement UI Button to return to GameScene
11. Inside LoseScene, create a LoseSceneController script, and implement UI Button to return to GameScene
12. Incorporate character animation (idle and walk)
13. Build an executable file

Submission

Folder structure
00_HanselKoh_GameManager

- Source
- Build