2D Game Design in Unity

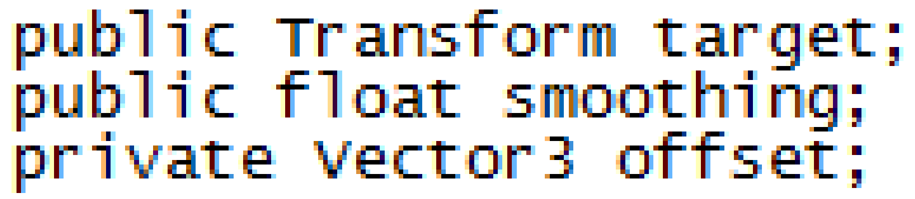
Lab 6 – Keeping Score + Camera Follow + Parallax Backgrounds

Exercise #17 - Camera Follow

- To make the camera follow your player throughout the level, you must create a script and attach it to your Main Camera
- Create **New Component > New Script > Name it CameraFollow** or any suitable name
- Attach CameraController to Main Camera
- Open script in Visual Studio

Exercise #17 - Camera Follow (Cont.)

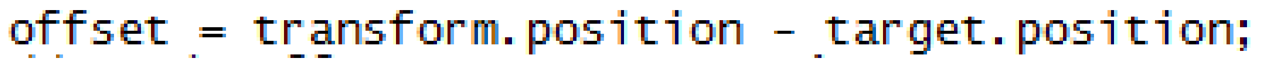
- Create a public variable of type Transform. This will be the object the camera must follow.
  - Reminder: any GameObject in your game **must** have a Transform component. It contains the rotation, position and scaling data of that game object.
- Create a public float and name it smoothing. This value controls how smoothly your camera follows the player game object. If your camera's movement is jittery or too slow, you may need to play with the value
- Create a private variable of type Vector3. This value controls how much space the camera leaves in front of the player, and how much it leaves behind him



```
public Transform target;
public float smoothing;
private Vector3 offset;
```

Exercise #17 - Camera Follow (Cont.)

- In the Start() function, add

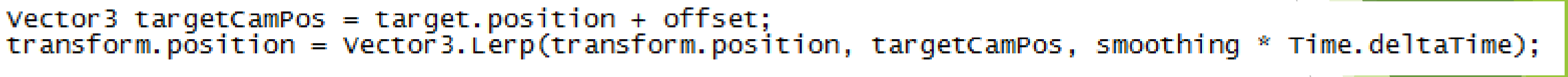


```
offset = transform.position - target.position;
```

  - The offset is value that the camera must allow so that we can see a bit of what's in front of the player and what's behind him, and it's calculated by subtracting the *position of the player from the position of the camera*

Exercise #17 - Camera Follow (Cont.)

- In the FixedUpdate function, add()



```
Vector3 targetCamPos = target.position + offset;
transform.position = Vector3.Lerp(transform.position, targetCamPos, smoothing * Time.deltaTime);
```

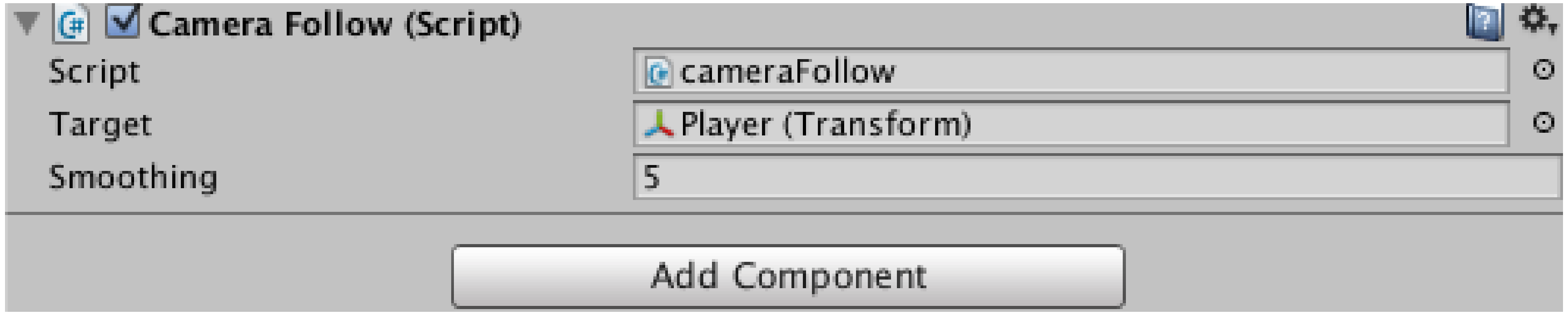
  - The targetCamPos is the variable that stores where the camera should actually be when it follows the player every frame. Its value is the position of the player plus the offset (space left by camera so player can see ahead and behind)
  - The Lerp function stands for Linear Interpolation. It will help us calculate how fast the camera will move, and how it will follow the player game object. The camera’s position is determined by taking its previous position, target position, and speed multiplied by the time passed since the last frame

Note on Lerp Function

- One way to explain Linear Interpolation is that it is a mathematical formula designed to take **3 values**: the FROM/BEGINNING point, the TO/ENDING point, and the smoothness by which you arrive to the ending point
- Example: If you wish to move from point 1 to point 3 on the x-axis, you provide that the FROM value is 1, the TO value is 3, and the smoothness could be 0.1. As you move between points 1 and 3, you'll automatically pass through points 1.1, 1.2, 1.3…2.2,2.3…etc. until you arrive at 3.

Exercise #17 - Camera Follow (Cont.)

- Go back to Unity
- Give your target variable the player game object by dragging it from the Hierarchy window, and experiment with a smoothing value of 5. Then run your scene



Camera Follow (Script)

Script	cameraFollow
Target	Player (Transform)
Smoothing	5

- If your camera moves smoothly, but your player game object doesn’t, go to its Rigidbody2D component and turn on Interpolation.
- (Your camera movement may be a bit laggy anyway if the computer’s graphics card is limited)

Note on Camera Control

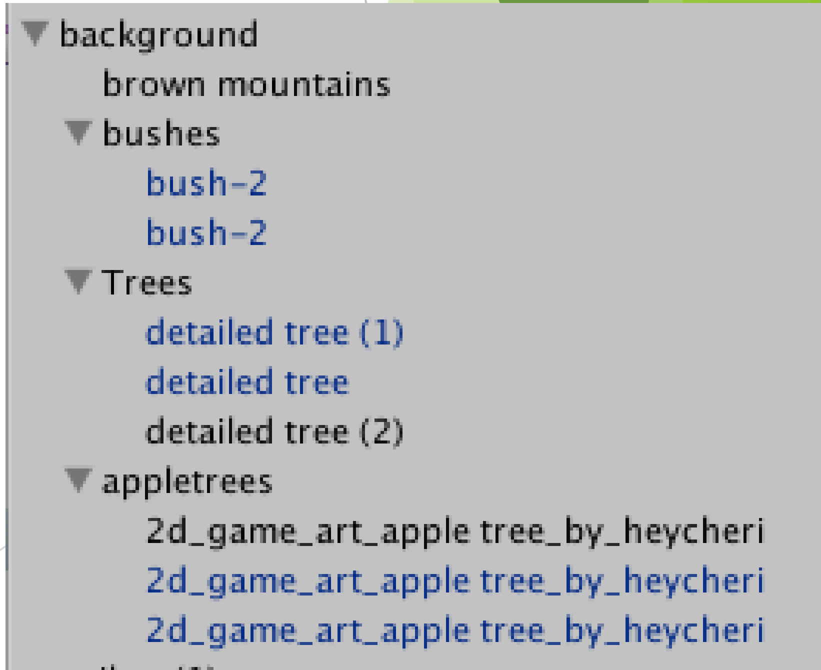
- There are many ways to control camera movement using script. This is only the easiest way. If you need to make it more tailored to your game, be sure to check tutorials on Camera Follow in Unity 2D on Youtube

Parallax Effect

- To give your game more visual depth, you can create what is called a Parallax Effect
- This effect makes your game's background feel like a car ride: Elements in the far horizon seem to move very slowly, while closer elements such as trees move by much faster
- Parallax effect can be written in many different ways. The following exercise #18 is one variation. Create a new script, name it BackgroundParallax and attach it to the Main Camera
  - Note: make sure you don't leave spaces in a script file's name

Exercise #18 – Parallax Effect

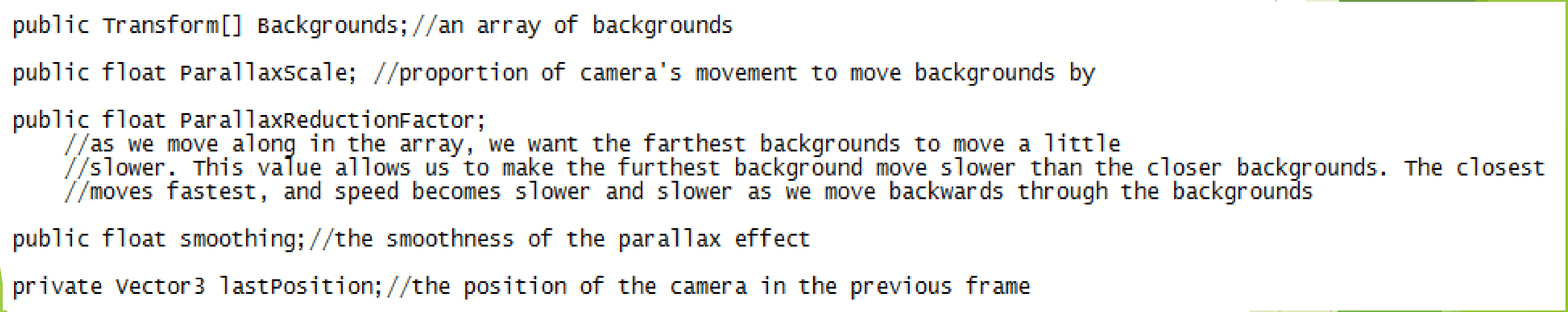
- Design your game's background however you want, and make sure all the elements are on the Background Layer
- Create a GameObject and call it Background. This will be the parent that contains all the background elements
- Inside Background, make more divisions. For example, you can have brown mountains at the very back, green mountains in the middle, and trees at the front



```
▼ background
    brown mountains
  ▼ bushes
      bush-2
      bush-2
  ▼ Trees
      detailed tree (1)
      detailed tree
      detailed tree (2)
  ▼ appletrees
      2d_game_art_apple tree_by_heycheri
      2d_game_art_apple tree_by_heycheri
      2d_game_art_apple tree_by_heycheri
```

Exercise #18 – Parallax Effect (Cont.)

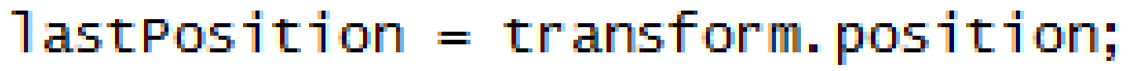
- After you've designed your background, go to your Parallax script and open in Visual Studio. Declare the following variables



```
public Transform[] Backgrounds;//an array of backgrounds

public float ParallaxScale; //proportion of camera's movement to move backgrounds by

public float ParallaxReductionFactor;
    //as we move along in the array, we want the farthest backgrounds to move a little
    //slower. This value allows us to make the furthest background move slower than the closer backgrounds. The closest
    //moves fastest, and speed becomes slower and slower as we move backwards through the backgrounds

public float smoothing;//the smoothness of the parallax effect

private Vector3 lastPosition;//the position of the camera in the previous frame
```

Exercise #18 – Parallax Effect (Cont.)

- In the Start() function, add:

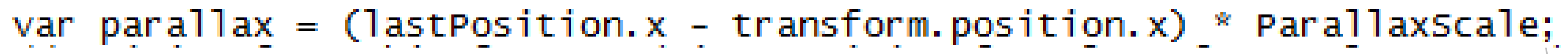


```
lastPosition = transform.position;
```

- This initializes your camera's position. Unity does not know where exactly your camera is when this script starts executing, so the camera's position from the last frame is assigned to be the camera's position in the current frame

Exercise #18 – Parallax Effect (Cont.)

- In the Update() function, add:

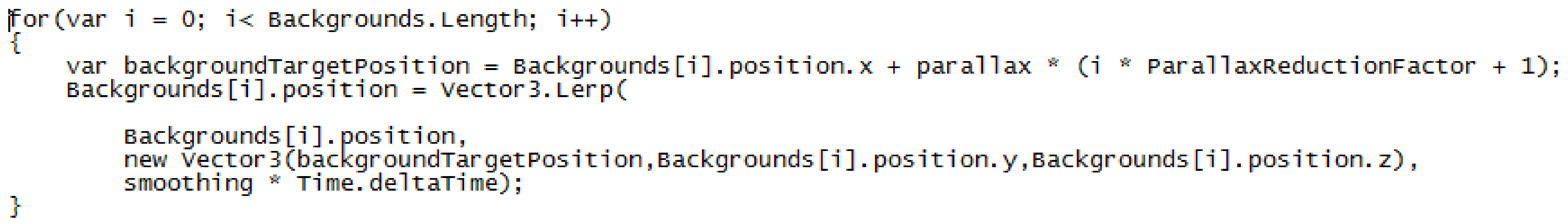


```
var parallax = (lastPosition.x - transform.position.x) * ParallaxScale;
```

- Create a var named parallax. This variable will contain the difference between the camera's position from this frame and its position from last frame along x-axis, and multiply by the ParallaxScale. This gives you the overall speed the camera is supposed to move.

Exercise #18 – Parallax Effect (Cont.)

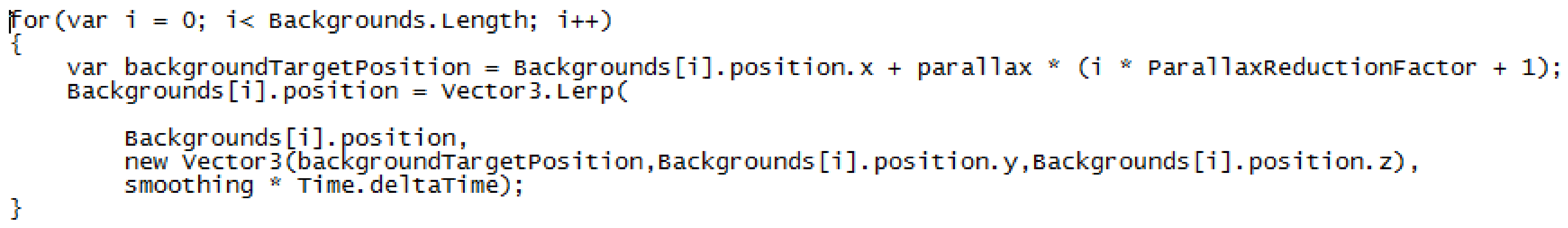
- Create a loop that iterates through the backgrounds array



```
for(var i = 0; i< Backgrounds.Length; i++)
{
    var backgroundTargetPosition = Backgrounds[i].position.x + parallax * (i * ParallaxReductionFactor + 1);
    Backgrounds[i].position = Vector3.Lerp(

        Backgrounds[i].position,
        new Vector3(backgroundTargetPosition,Backgrounds[i].position.y,Backgrounds[i].position.z),
        smoothing * Time.deltaTime);
}
```

- create a var and name it backgroundTargetPosition. This variable contains the location the background is supposed to be at the next frame (along x-axis). Assign it the value of the current x-position of the background layer you are currently at, and add the parallax factor (the camera position at that particular moment). Multiply the current iteration (background layer) by the Parallax Scale Factor
- Inside the loop, we need a statement that makes every layer of background slower than the layer before it.

Exercise #18 – Parallax Effect (Cont.)



```
for(var i = 0; i< Backgrounds.Length; i++)
{
    var backgroundTargetPosition = Backgrounds[i].position.x + parallax * (i * ParallaxReductionFactor + 1);
    Backgrounds[i].position = Vector3.Lerp(

        Backgrounds[i].position,
        new Vector3(backgroundTargetPosition,Backgrounds[i].position.y,Backgrounds[i].position.z),
        smoothing * Time.deltaTime);
}
```

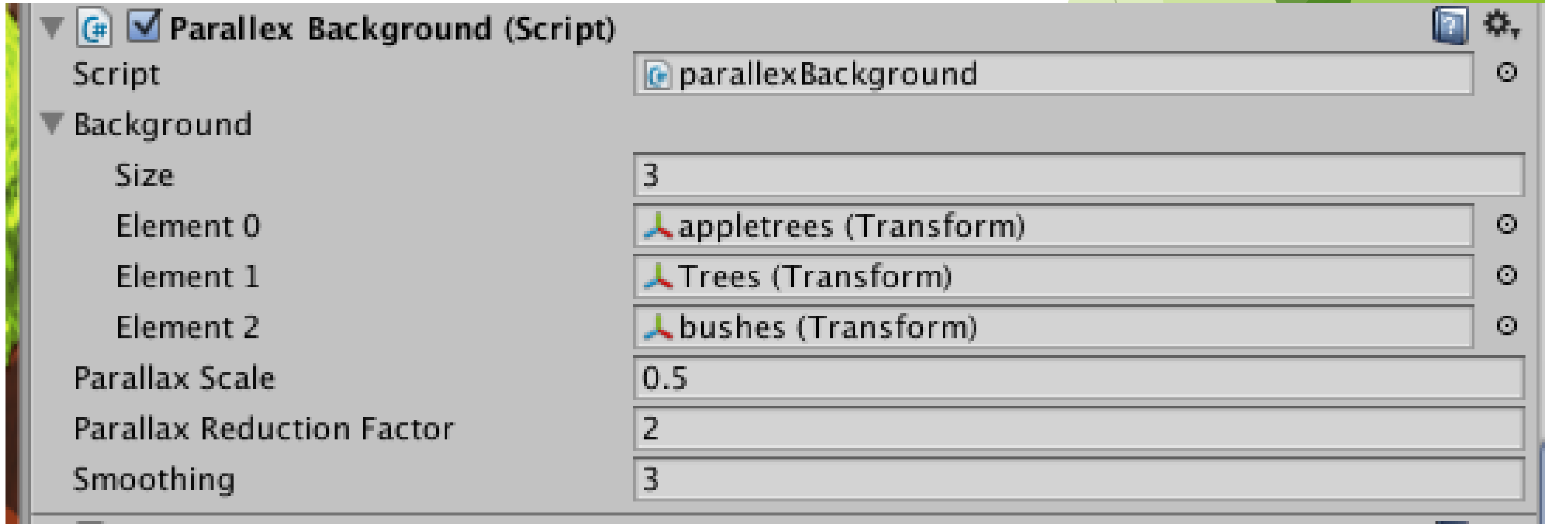
- Use the Lerp function to determine the FROM value as the current background layer’s position, The TO value which is where the background is supposed to be (note that it takes 3 values because it’s a vector3 type), and the smoothing factor multiplied by the time that passes between each frame and the next when the game runs

Exercise #18 – Parallax Effect (Cont.)

- Go back into Unity and see the public variables appear as the script's properties
- Drag the Background super parent from the Hierarchy and drop it onto the Background property in the Inspector as seen below
- Drag each of the 3 background children and drop them into the text boxes as seen below
- Experiment with the values of the

Parallax Scale, Reduction Factor, and

Smoothing

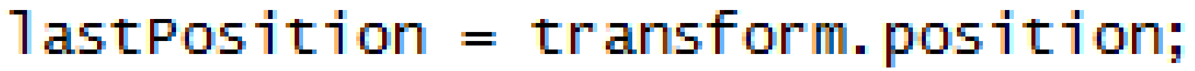


Parallex Background (Script)

Script	parallexBackground
Background	
Size	3
Element 0	appletrees (Transform)
Element 1	Trees (Transform)
Element 2	bushes (Transform)
Parallax Scale	0.5
Parallax Reduction Factor	2
Smoothing	3

Exercise #18 – Parallax Effect (Cont.)

- After the loop ends, do not forget to update the camera’s position by adding once more the statement:



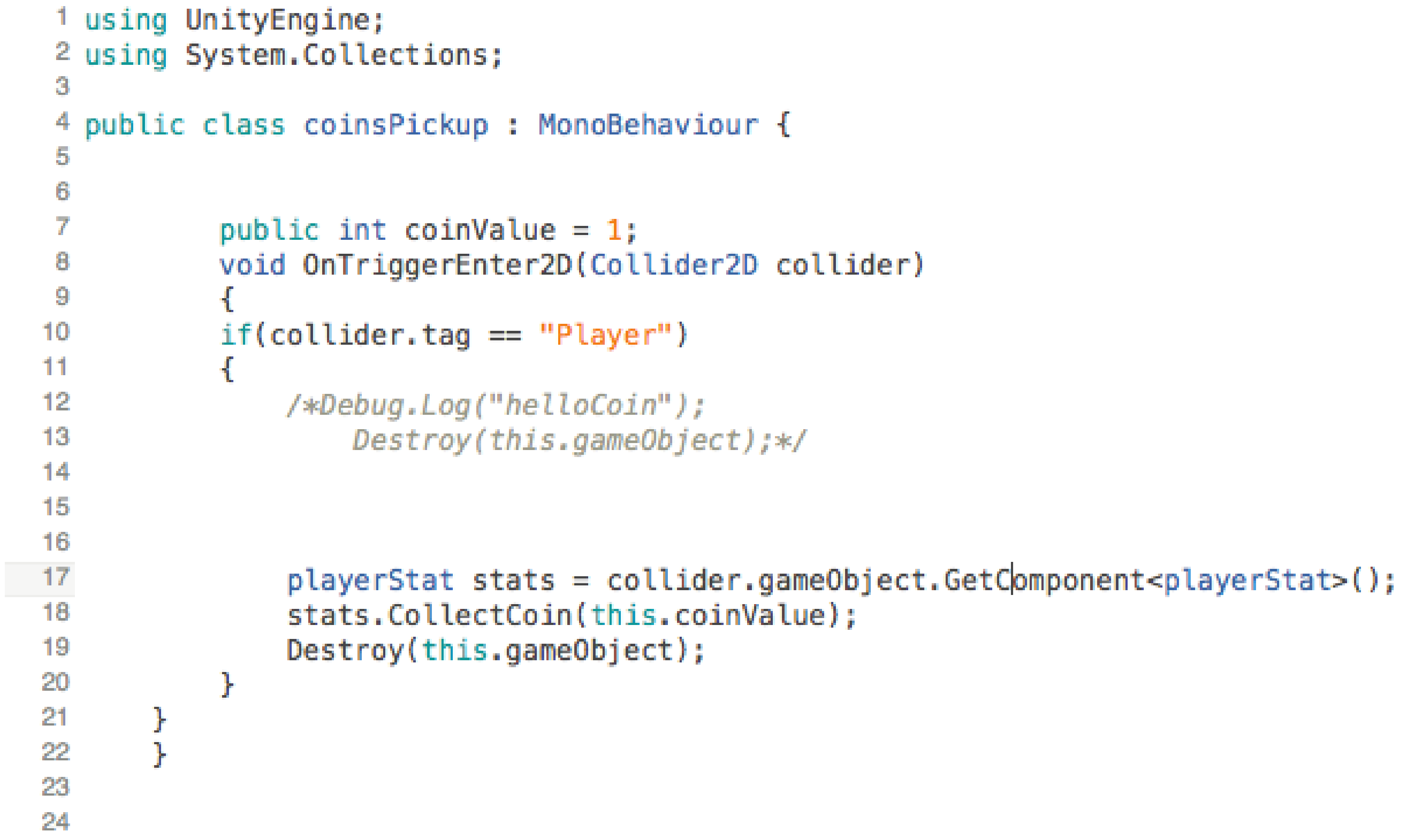
```
lastPosition = transform.position;
```

- So when the Update function is called against for the next frame, the camera is where it should be
- Run your scene

Exercise # - Collect Coins/Items

- First, create coins:
- Choose suitable coin sprites and make them varied – gold, silver and bronze coins. Each will have a different value when the player picks them up
- Add a Circle Collider 2D to each coin
- Check the “Is Trigger” property
- Create a new script and name it CoinPickup

Exercise # - Collect Coins/Items (Cont.)



```
using UnityEngine;
using System.Collections;

public class coinsPickup : MonoBehaviour {


        public int coinValue = 1;
        void OnTriggerEnter2D(Collider2D collider)
        {
        if(collider.tag == "Player")
        {
            /*Debug.Log("helloCoin");
                Destroy(this.gameObject);*/



            playerStat stats = collider.gameObject.GetComponent<playerStat>();
            stats.CollectCoin(this.coinValue);
            Destroy(this.gameObject);
        }
    }
    }


```

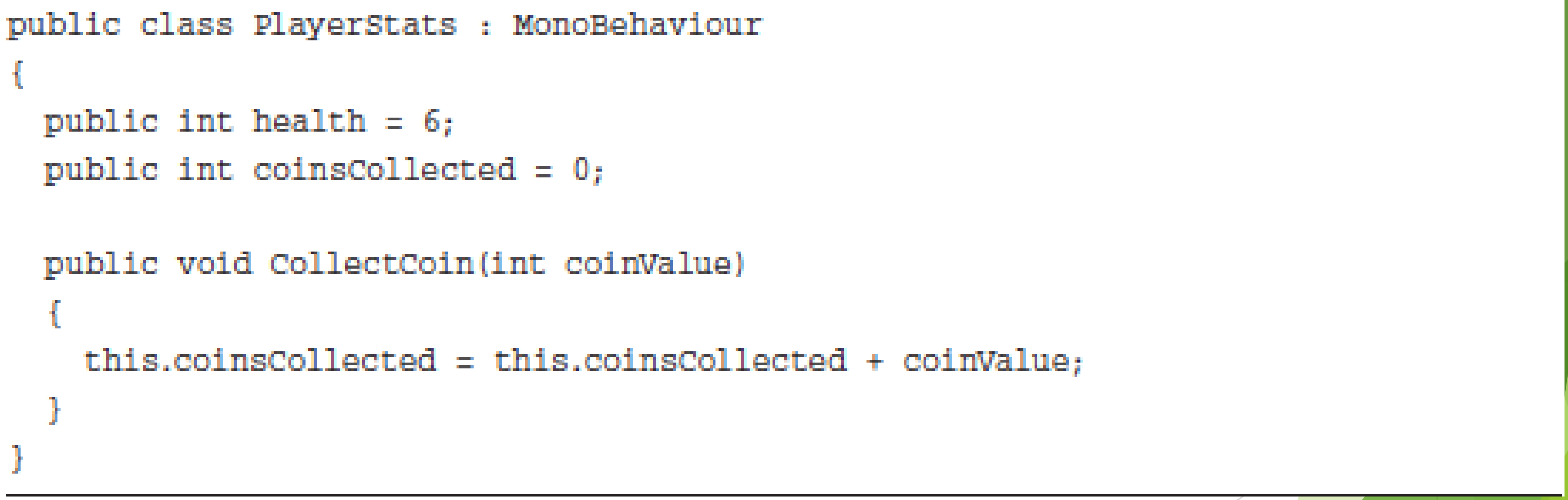
Exercise # - Collect Coins/Items (Cont.)

- The script executes when the player’s collider collides with the coin’s collider due to the checked IsTrigger property
- The Destroy function makes the coin object disappear off the screen, giving the illusion that the player has picked it up
- The public variable also allows to set the value of the coin . Set the values for the coins as follows:
  - coinBronze: Set the “Coin Value” property to 1.
  - coinSilver: Set the “Coin Value” property to 5.
  - coinGold: Set the “Coin Value” property to 10.

Exercise # - Collect Coins/Items (Cont.)

- You have the player collecting coins, but the total number of coins are not stored yet
- A PlayerStats Component will be added to the Player to store information about the number of coins the player has collected .In order to do that ,create a new script named “PlayerStats” and add it to the Player GameObject

Exercise # - Collect Coins/Items (Cont.)



```
public class PlayerStats : MonoBehaviour
{
  public int health = 6;
  public int coinsCollected = 0;

  public void CollectCoin(int coinValue)
  {
    this.coinsCollected = this.coinsCollected + coinValue;
  }
}
```

References:

- Shooting Projectiles & Camera Control - Unity 2D Platformer Tutorial - Part 8. URL retrieved from: https://www.youtube.com/watch?v=8aVZuL9ocrk
- Creating 2D Games in Unity 4.5 #15 - Camera Controller. URL retrieved from: https://www.youtube.com/watch?v=u67fbxe8xxY
- 2D RPG Smooth Camera - Unity3D. URL retrieved from: https://www.youtube.com/watch?v=KMhPYf9zzlA
- How to make a 2D Platformer - Parallax Scrolling - Unity Tutorial. URL Retrieved from: https://www.youtube.com/watch?v=5E5_Fquw7BM
- Unity 5 2D Platformer Tutorial - Part 8 - Camera Bounds. URL Retrieved from: https://www.youtube.com/watch?v=I6xmOMsRWeo