```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class GameController : MonoBehaviour
7  {
8      // Attack Prefabs
9      public GameObject squareAttack;
10     public GameObject horizontalAttack;
11
12     // Instance of the Game Over script
13     public GameOverScript gameOver;
14
15     // Variables related to attack spawning
16     public float spawnAttackRepeat;        // Time delay between attack spawns
17     public float phaseGap;                 // Time delay between phases
18     public bool gameRunning = false;       // Is the game running
19     public int score = 0;                  // To keep track of the score
20     public int phaseSize = 5;              // How many attacks in each phase
21
22     // Variables related to tutorial
23     public bool tutorialPhase = false;     // Is it the tutorial phase
24     public float tutorialStartTimeBuffer;  // Time delay between start of game
          and start of tutorial phase
25     public float tutorialReadTime;         // How long to leave up each
         tutorial text
26     public List<Text> tutorialTexts;       // List of tutorial texts to
         display
27
28     // Player game object
29     GameObject player;
30
31     // Player's transform (could be gotten from player game object but
32     // doing this makes lines shorter and easier to read)
33     Transform playerTrans;
34
35     // Instance of generate phase script
36     GeneratePhase phaseGenerator;
37
38     // Start is called before the first frame update
39     void Start()
40     {
41         // Populate variables with instances of objects
42         player = GameObject.FindGameObjectWithTag("Player");
43         playerTrans = player.GetComponent<Transform>();
44         phaseGenerator = GetComponent<GeneratePhase>();
45         gameOver = GetComponent<GameOverScript>();
46
```

```
47          // Start the tutorial phase and make sure all tutorial texts are
              hidden
48          tutorialPhase = true;
49          foreach (Text tutText in tutorialTexts) {
50              tutText.enabled = false;
51          }
52
53          // If statement used in testing to skip the tutorial
54          if (gameRunning) {
55
56              // Start the game
57              StartCoroutine(ExecutePhase(phaseGenerator.GenerateNewPhase(score,
                  phaseSize)));
58
59          } else if (tutorialPhase) {
60
61              // Start the tutorial
62              StartCoroutine(DoTutorial());
63          }
64      }
65
66
67      IEnumerator ExecutePhase(List<string> nextPhase) {
68
69          // For each attack in the next phase string
70          foreach (string attack in nextPhase) {
71
72              // Wait the spawn delay
73              yield return new WaitForSeconds(spawnAttackRepeat);
74
75              // Make sure the game is still running
76              if (gameRunning) {
77                  if (attack == "square") {
78
79                      // Spawn in square attack at player's position
80                      Instantiate(squareAttack, new Vector3
                        (playerTrans.position.x, playerTrans.position.y, 10),
                        squareAttack.transform.rotation);
81
82                  } else if (attack == "horizontal") {
83
84                      // Spawn in horizontal attack at player's y position
85                      Instantiate(horizontalAttack, new Vector3(0,
                        playerTrans.position.y, 10),
                        horizontalAttack.transform.rotation);
86
87                  } else {
88
89                      // I love self depricating error messages, it really makes
```

```
                    me feel responsible for whatever wrong I did
90                  Debug.Log("You messed up the if statement for the attack
                    types retard. What kind of an attack is " + attack + "?");
91              }
92          }
93      }
94
95      // At the end of the phase, make sure the game is still running
96      if (gameRunning) {
97
98          // Add 1 to the score
99          score += 1;
100
101         // Wait for the phase gap
102         yield return new WaitForSeconds(phaseGap);
103
104         // Execute the next phase
105         StartCoroutine(ExecutePhase(phaseGenerator.GenerateNewPhase(score,
               phaseSize)));
106     }
107 }
108
109 // Ends the game
110 public void EndGame() {
111
112     // Sets game running to false
113     gameRunning = false;
114
115     // Puts all on screen attacks in an array by searching for all gameo
          bjects with the tag "Attack"
116     GameObject[] attacksOnScreen = GameObject.FindGameObjectsWithTag
          ("Attack");
117
118     // Go through each one and delete them
119     foreach (GameObject attack in attacksOnScreen) {
120         Destroy(attack);
121     }
122
123     // Run game over function in game over script (its handles the UI
          stuff)
124     gameOver.GameOver();
125 }
126
127
128 // Do the tutorial
129 IEnumerator DoTutorial() {
130
131     // Wait the start time buffer
132     yield return new WaitForSeconds(tutorialStartTimeBuffer);
```

```
133
134         // For every tutorial text
135         foreach (Text tutText in tutorialTexts) {
136
137             // Show it on screen, wait the read time, then hide it again
138             tutText.enabled = true;
139             yield return new WaitForSeconds(tutorialReadTime);
140             tutText.enabled = false;
141         }
142
143         // Set game running to true and tutorial phase to false
144         gameRunning = true;
145         tutorialPhase = false;
146
147         // Start the game
148         StartCoroutine(ExecutePhase(phaseGenerator.GenerateNewPhase(score,
              phaseSize)));
149
150     }
151 }
152
```