***Introduction to Graphics and Mobile Gaming***

**LAB 4: Part 3**

**Game features**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc36131001)

[2 Scoring system 1](#_Toc36131002)

[2.1 Inserting new element 1](#_Toc36131003)

[2.2 Scoring mechanism 1](#_Toc36131004)

[2.2.1 Code indications 2](#_Toc36131005)

[2.2.2 Scene elements 3](#_Toc36131006)

[3 Adding a timer 3](#_Toc36131007)

[4 Creating a menu 5](#_Toc36131008)

[5 Exit game 6](#_Toc36131009)

# Introduction

In this lab, we will look at adding more features such as a main menu, a scoring system and modifying our existing assets even further. By the end of this lab, we should have a fully featured gameplay experience with gameplay features, scoring, multiple scenes, and camera controls.

# Scoring system

## Inserting new element

The first thing we should do is add a feature asset to our scene to act as a centerpiece of the design

* Click on Assets and then advanced>meshes. We will use one of the meshes we haven’t used yet, say the Pheonix. Select it and place it in the middle of the room. You might wish to scale it by 10 in all dimensions.
* Now go to advanced>materials and place the pheonix2 material on the phoenix statue.

We will now modify the explosion controller so that the sparks are emitted only upon collision with the new object. Simply take the script and *explosionMobile* from the chess pieces and apply them both to the phoenix instead. Remember to attach the script to the phoenix element; and the Spark only to the Transform field of the field; you can also edit the Sparks to 0.5 duration and Loop.

Make sure you remember to put a RigidBody and a Box collider around the phoenix so collision gets recorded.

Try to throw a chess piece to the target; we should see sparks upon collision with the phoenix.

You might notice that when you hit the phoenix with a piece, it also moves; but we’d like it to be fixed in the center of the room. Tick the ***Is Kinematic*** box in the Inspector tab.

## Scoring mechanism

Next, we want to combine the collision mechanic we have just created with a scoring mechanic. Meaning every time a chess piece hits the phoenix, we want to score a point. For this, we will use a C# script. Open the scripts folder and create a new script by right-clicking and then selecting create and then selecting C# script. Name the new script ***scoringOnCollision***. We will attach this script to the phoenix object.

We already know how to detect when two objects collide, as this was implemented in the explosionController script. Once the collision is detected, the code will need to determine whether it came from the chess piece or another object.

Let’s write a function in the *explosionController,* which uses Debug.Log("Touched dynObj") to let us know if everything is working; we will see the message during run time in the Console panel. Attach the script to the target.

Remember that LayerMask.NameToPlayer(“layer”) will return an integer of the layer that you are referring to.

The solution to this step is in this lab folder, if necessary.

### Code indications

We will set up a variable that increases every time our chess piece hits another *dynObj*. To do this, set up a new variable and print it to the Console every time collision is triggered. What we will want to do is take this variable and display it on the screen. To do this, there are some things we need to add.

* Firstly, we need to add the UI library to the top of our code, by using this line:

using UnityEngine.UI;

* Then, we add the text element; a text element is a part of the UI library. We add two after the declaration of the class:

public Text countText;  
public Text winText;

* First, we’re going to deal with the **countText** value first. We do this by adding the following function to our code:

public int pointsLimit = 5;

void SetCountText ()  
    {  
        countText.text = "Count: " + numberOfPoints.ToString ();  
        if (numberOfPoints >= pointsLimit)  
        {  
            winText.text = "You  Win!";  
        }  
    }

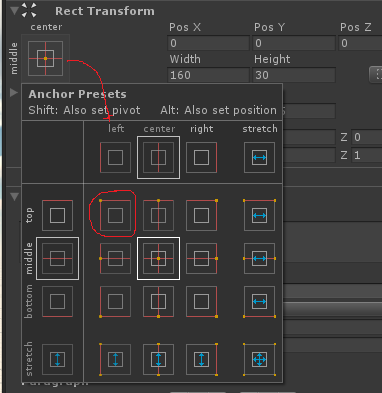
* At the start of the gameplay, we will have to reset the text UI elements, so make sure we add the following to the start function:

SetCountText ();  
 winText.text = "";

Finally, make sure that the *SetCountText* is called after increasing the number of points in the collision function.

### Scene elements

1. **Score**

Now back in our scene, add a new canvas as we did with the Joystick and rename it as GameUI. Add it as a child of a Text object, go to UI, and then select text. Rename the text element to “score”; the *text* label will appear in the middle center

With the text element selected, go to the inspector panel and click on the red pointer rectangle, and then by pressing alt+shift, select the one with the pivot in the top left corner (see image).

Then, adjust the X coordinate to 10, and Y to −10. Increase the font size to 25 so that it is more visible. Then, change the colour to something that is a little more visible in our scene, such as green. Be sure to check the new settings by running the scene.

Finally, attach the ***score*** element to the new script ***scoreOnCollision***. Check the scene again and try scoring; hopefully, we should see the points count increase with each successful collision.

1. **Winning text**

We should conduct a similar process for our winning text message, this time placing it in a different position. Attach the text elements to the script properly as well. Set the minimum score to 5 so that the player wins the game upon getting a score of 5.

As we can see, the Win message does not disappear yet.

We need to set a delay and clear the message. This can be done in the same exact way the explosionController script stops the particles when the object is not colliding with anything. Clear the score as well and bring it back to 0.

Also reset the points to 0 once the game has been won.

# Adding a timer

To add a layer of difficulty to our game, we want to add a time limit so that the player loses the game when the timer expires.

1. To add a **timer**, let’s add a new text field in our code for our timer:

public Text timeText;

* Then, add a float to store the maximum time (20 seconds). Name this float **timeLeft.**
* Then, in the start function, initialize the text on screen by writing:

timeText.text = "Time: " + ((int)timeLeft).ToString ();

* Finally, inside the update function, decrease the time left and as done in start print the time, this will have to be casted to integer or it will not appear on screen (it will appear too fast and we won’t be able to see it).

To decrease the time, we can use the following line:

timeLeft -= Time.deltaTime;

* Create a new UI text element, rename it “time,” and attach it to the ScoringOnCollision script from the target element inspector. Make sure the text is in a visible position, place it as done for the count text.
* Try to play, and we should see the time display on screen! It should stop when zero is reached!

The solution for this is in the code folder for this lab.

1. The next step is to make sure that if the time ends before the user has won the game, the game stops and the user is informed that it’s **gameOver**.

* Add a text UI element, name it GameOver, and place it at the center of the screen.
* Head back to our code and add a public Text variable, save this and drag the text UI element GameOver from the hierarchy panel to the field of the script attached the to target. Press Play to ensure that the field has appeared, which will produce an error. Stop the game, and the new text field should be in place.
* Set the text to be empty in the *start()* function and then create a new function that displays a game over message.
* In the update function, create some code that will check to see if the time is smaller than 0, and if so, call the *game over function*.
* We also need to make sure when the game is over. To do this, add a bool variable and name it *endgame*. Initialize the variable to *false* in the *start()* function and set it to *true* in our *game over* function.
* We need to conduct the same process for *SetCountText* function and specify it to return *true* if the score is higher than the *pointLimit*.
* The update function also needs to be modified so that its content is only called if *endgame* is *false*.
* This also needs to be completed for the onCollisionEnter function.

Test the game on our device, build the apk, and install it.

# Creating a menu

Next, we will create a menu from which the user can access the game.

* Click on File -> New Scene (the next time we will switch scene Unity will ask to save it, remember to name it “menu”). Click on the 2D button to flatten the scene and see it better.
* Drag a new skybox into this scene using the method we used previously for skyboxes.
* Then, click on Game Object and then UI and then button. Place the button at the top-center of the screen and resize it to that it is of a good size for the window.
* Rename the element *game*, then from the Hierarchy panel, click the small gray arrow, then click on the text option, and from the inspector, change the text to “GAME.”
* Open the scripts folder and create a new script; we will call this *menuControl*. Open the script and paste this code (substituting the existing one):

using UnityEngine;  
using System.Collections;  
using UnityEngine.SceneManagement;  
  
public class menuControl : MonoBehaviour {  
  
    public void changeScene(string scene){  
  
        SceneManager.LoadScene(scene);  
    }  
}

* A screenshot of a cell phone

  Description automatically generatedDrag and drop the new script into the Canvas Inspector and then select the button game from the Hierarchy panel. Click the + symbol on the On Click() section at the end of the Button Script (we can find this in the inspector panel). In the “None” field, drag and drop the Canvas element from the hierarchy panel here; then, on the drop-down menu displaying “No Function,” select menuControl and then changeScene(string). Underneath, we should see an empty field, type the name of main game’s scene.
* Open the Build Settings in the top tab (scenes in build), add the menu scene, and then proceed and add the main scene of the game.

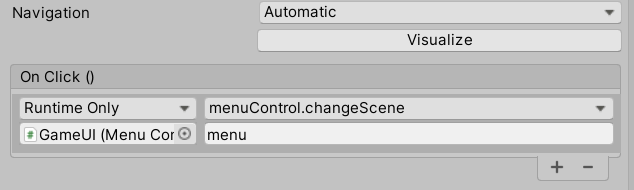
Now return to your menu scene and press Play. You should now be able to click on the button, and it will open the main game scene.

# Exit game

Our last task is to enable the user to get back to the main menu if any of the win or lose conditions are met. We also want to allow them to exit the game if they so choose to get back to the main scene.

To exit the game, add a button and name it *exit* (add it to the main game scene just like previously).

Place the button in the upper corner of the main game scene to avoid any conflicts when joysticks are being pressed

* Drag the *menuControl* script onto the Canvas inspector. Then, select the button and add the On Click functions as described previously for the game button. This time, we want to redirect the user to the menu scene so type “menu” in the empty label that defines the scene to be redirected to.

Now build the project and install the apk on your device. Test the different functions and effects and spot any other additions or changes that could be made.

This is the end of this lab; we now have the basis of a game framework. In the next lab, we will be looking at adding VR functionality to our game.