|  |  |
| --- | --- |
| ITB logo portrait B&W | INSTITUTE OF TECHNOLOGY BLANCHARDSTOWN  A Taster of Computing  [[VERSION – Unity 2D – C# language]] |

Gravity Guy 2D (2015) - a little computer game...

Part 3 – improving the ‘view’ (UI display of score) …



Welcome to “Gravity Guy”. In this multimedia programming exercise you will create a little 2D computer game.

CONTENTS

1 Recap, and aims of this part of the tutorial 3

2 Play a sound when a piece of food is eaten 4

3 Respawn the hero character when it falls down too low … 8

4 Separate the GUI ‘view’ from the Player ‘model’ 9

5 Add killer ‘spike’ objects to the scene 11

6 Add a ‘game over’ scene to our project (for when all lives lost) 14

7 Congratulations … 21

# Aims of this part of the tutorial

## New features / skills to be learned in this part of the tutorial

In this part of the tutorial you will add the following features to our game:

* Play a sound each time we add to the score
* Respawn the user when they fall too far down the screen
* Separate the score ‘view’ display to the user, from the observed score property inside the Player object
* Add killer ‘spike’ gameObjects to the scene
  + And have these make the player lose lives and respawn each time they are hit
  + This will involve adding a ‘lives’ property to our Player, and adding corresponding UI display of this property
* Add a “Game Over” scene to our game
  + And have the user see this scene when they lose their last life

# Separate the GUI ‘view’ from the Player ‘model’

## Separate View code from our Player behaviour script class

As a game becomes more complex, it becomes important to ensure each script class has a well defined boundary of responsibility. One example of this is our Player script class – it should maintain and update important properties of our player, such as the player’s SCORE and number of LIVES left, and any INVENTORY the player may be carrying.

However, the Player script class should NOT worry about HOW the values of these properties are COMMUNICATED to the user.

Each important property should be a PRIVATE variable, but should offer PUBLIC accessor methods (getters and setters) for any other objects that have responsibilities that require them to access these properties of the Player.

We are now going to REFACTOR our code, to separate the VIEW of player properties (our GIU to the user) from the Player script class itself.

## Remove GUI code from Player, but add public GET method for ‘score’

Let’s remove the GUI code from our Player script class, but add a public GETTER method for the important ‘score’ property:

* In the **Project** panel select the **Scripts** folder
* Double click the **Player** script class file to load it into the **Monodevelop** editor
* Edit the code as follows:
  + Remove method OnGUI()
  + Add a new method GetScore()

*using UnityEngine;  
using System.Collections;  
  
public class Player : MonoBehaviour*

*{  
    private int score = 0;*    public int GetScore()

{  
        return score;  
    }  
  
    private float deathY = -5f;  
  
    private void Update(){  
        float y = transform.position.y;  
  
        if(y < deathY){  
            MoveToStartPosition();  
        }  
    }

private void MoveToStartPosition()

{

            Vector3 startPosition = new Vector3(0,5,0);  
            transform.position = startPosition;

}

*private void OnTriggerEnter2D(Collider2D c)*

*{  
        string tag = c.tag;  
  
        if("Food" == tag)*

*{  
            score++;  
            audio.Play();  
        }  
    }  
}*

## Create new GameGUI script class, and add it to the Main Camera

Create a new C# script named GameGUI:

* In the **Project** panel select the **Scripts** folder
* From the **Project** ‘Create’ menu choose ‘C# Script’ and rename this new script “GameGUI”
* Double click your new **GameGUI** script class file, and in MonoDevelop edit its code to be the following:

**using UnityEngine;  
using System.Collections;  
  
public class GameGUI : MonoBehaviour**

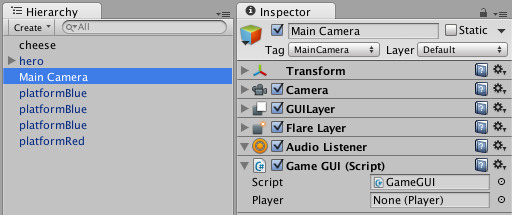
**{  
    public Player player;  
  
    private void OnGUI()**

**{  
        int playerScore = player.GetScore();  
        string scoreMessage = "Score = " + playerScore;  
        GUILayout.Label(scoreMessage);  
    }  
}**

## Add an instance of our GameGUI script class as a component of Main Camera

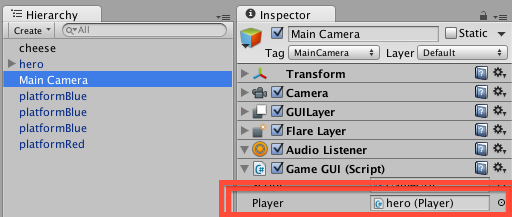
A script class does nothing, unless an instance of it has been added as a component of a gameObject in the scene. Do the following:

* In the **Project** panel select the **Scripts** folder
* In the Hierarchy select the **Main Camera** gameObject
* Drag the C# script class file from the **Project Scripts** folder into the **Inspector**
* You should now see that Hiearchy gameObject **Main Camera** now has a component Game GUI (Script)



We now need to make a link between the **hero** gameObject in our Hierarchy, and the ‘player’ variable in our GameGUI component in the **Main Camera**. Do the following:

* In the Hierarchy select the **Main Camera** gameObject
* Drag the **hero** gameObject from the Hierarchy into the Player property of the Game GUI (Script) component of the **Main Camera**
* The public ‘Player’ variable of the GameGUI script component of Main Camera should now indicate that it is a reference (link) to **hero (Player)**
  + The instance of the Player script class that is a component inside the **hero** gameObject



The game will look just the same to the user – but you know that the GUI ‘view’ has now been separated from the Player ‘model’ of player’s properties.

# Add killer ‘spike’ objects to the scene

## Add a new property ‘lives’ to our Player script class

Let’s start the player off with 3 lives. We need to add a property ‘lives’ to our Player script class, and also add a public ‘getter’, to allow our GameGUI to retrieve and display the value of the ‘lives’ variable:

Add a ‘lives’ inteter property to script class Player, and a corresponding public ‘getter’:

* In the **Project** panel select the **Scripts** folder
* Double click the **Player** script class file to load it into the **Monodevelop** editor
* Edit the code as follows:
  + Add a new private integer property ‘lives’ initialised to 3
  + Add a new public method GetLives()

*using UnityEngine;  
using System.Collections;  
  
public class Player : MonoBehaviour*

*{  
    private int score = 0;***private int lives = 3;***public int GetScore()*

*{  
        return score;  
    }***public int GetLives()**

**{  
        return lives;  
    }**

## Edit our GameGUI to display the lives property to the user

Add code to our GameGUI script class to display the player’s lives integer on screen:

Add a ‘lives’ integer property to script class Player, and a corresponding public ‘getter’:

* In the **Project** panel select the **Scripts** folder
* Double click the **GameGUI** script class file to load it into the **Monodevelop** editor
* Edit the code as follows:
  + We will move the Score display code into new method DisplayScore()
  + We will create a new Lives display code method DisplayLives()

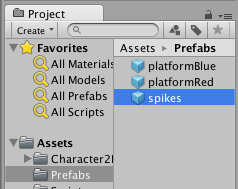
*using UnityEngine;  
using System.Collections;  
  
public class GameGUI : MonoBehaviour {  
    public Player player;***private void OnGUI(){  
        DisplayScore();  
        DisplayLives();  
    }  
      
    private void DisplayScore(){  
        int playerScore = player.GetScore();  
        string scoreMessage = "Score = " + playerScore;  
        GUILayout.Label(scoreMessage);  
    }  
      
    private void DisplayLives(){  
        int playerLives = player.GetLives();  
        string livesMessage = "Lives = " + playerLives;  
        GUILayout.Label(livesMessage);  
    }  
}**

## Add some copies of the ‘spikes’ prefab as gameObjects to the scene

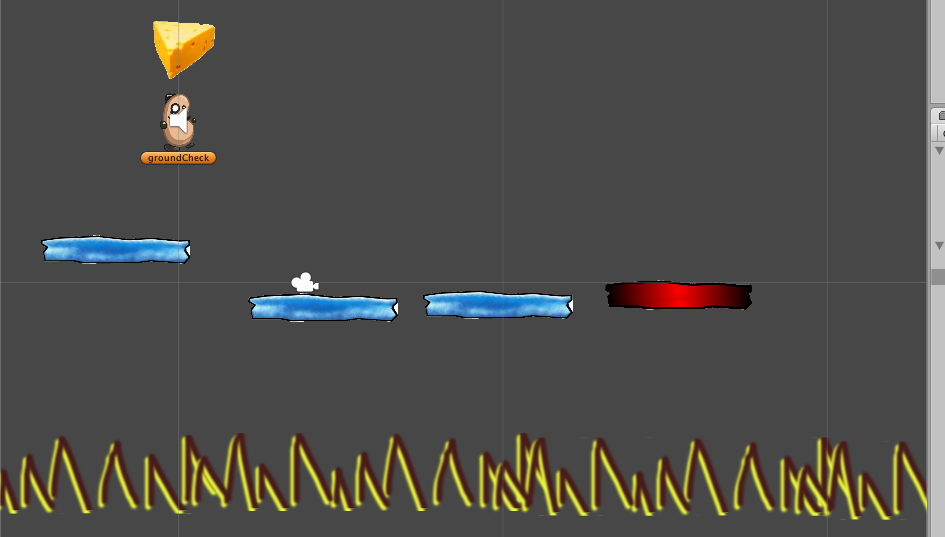
Rather than our player falling into some invisible minimum Y position, let’s actually display some ‘spikes’, whereby the player loses a life when they fall onto them.

Add some copies of the ‘spikes’ prefab as gameObjects to our scene, in a row BELOW the platforms:

* In the **Project** panel select the **Prefabs** folder
* Drag copies of the **spikes** prefab onto the scene, in a row below the platforms:

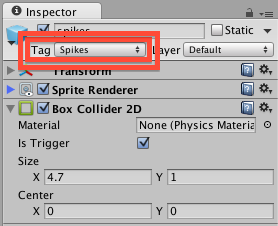


Your scene should now look something like the following:



­­­ ­­

If you select a ‘spikes’ gameObject, you’ll see that they have their collider ‘trigger’ ticked, and have the tag ‘Spikes’:



## Edit Player code, to decrement lives when hit something tagged ‘spikes’

We now need to edit our Player script class, so that when it collides with something tagged ‘spikes’, we decrement the lives, and move it back to the start position.

NOTE – we’ll also remove the code that repositions the character if the Y value is too low … so the onus is on the LEVEL DESIGNER to ensure all falls lead to ‘spikes’ …

Let’s edit our collision trigger method, and turn the code to move the object to the start position into a separate method:

* In the **Project** panel select the **Scripts** folder
* Double click the **Player** script class file to load it into the **Monodevelop** editor
* Edit the code as follows:

*using UnityEngine;  
using System.Collections;  
  
public class Player : MonoBehaviour*

*{  
    private int score = 0;  
    private int lives = 3;  
      
    public int GetScore()*

*{  
        return score;  
    }  
      
    public int GetLives()*

*{  
        return lives;  
    }*

*private void OnTriggerEnter2D(Collider2D c)*

*{  
        string tag = c.tag;  
  
        if("Food" == tag)*

*{  
            score++;  
            audio.Play();  
        }* **if("Spikes" == tag)**

**{  
            lives--;  
            MoveToStartPosition();  
        }  
    }  
  
    private void MoveToStartPosition()**

**{  
        Vector3 startPosition = new Vector3(0,5,0);  
        transform.position = startPosition;  
    }***}*

NOTE: Since out guy is being killed by the spikes, we have removed the deathY variable and the Update() method:

    private float deathY = -5;  
  
    private void Update()

{  
        float y = transform.position.y;  
  
        if(y < deathY)

{  
            MoveToStartPosition();  
        }  
    }

# Add a ‘game over’ scene to our project (for when all lives lost)

## Create a new blank scene, and name it “gameOver”

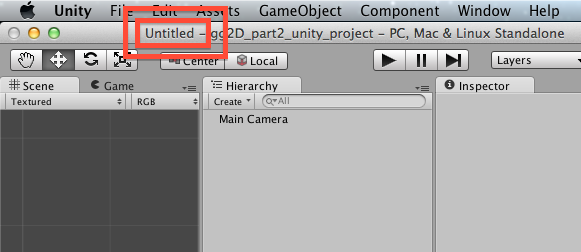
Interactive Multimedia games and applications tend to break up naturally into different screens, or levels, or ‘scenes’ as they are know in Unity:

* Welcome to the game
* Main Menu
* High Scores
* Credits
* Instructions
* Level 1 playing
* Level 1 complete (with button to start level 2)
* Leve 1 lost / Game Over
* Level 2 playing etc.

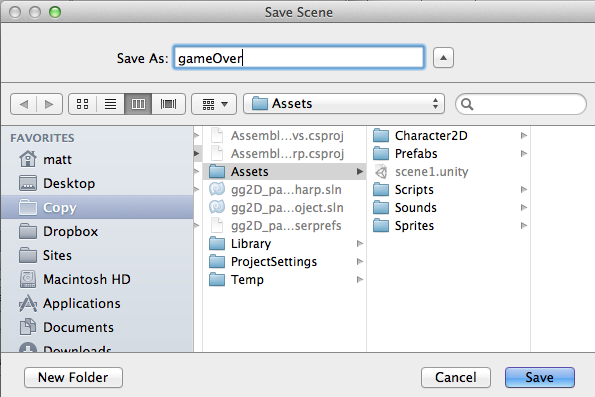
FIRST – save your work on **scene1** that you have been adding to up to know.

Create a new scene:

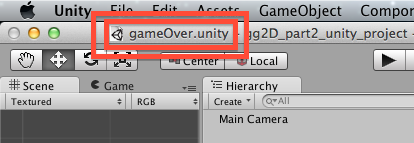
* Choose menu: **File | New Scene**
* Your should now see that all game objects in the Scene and Hiearchy panels have gone
  + Except for the default **Main Camera** that is always added to a new scene
* You can also see that the NAME of the scene you are now working on (in the Unity application window title bar) has changed from “scene1” to “untitled”



* Now SAVE and NAME your new scene
  + Choose menu: **File | Save Scene As …**
    - Save the scene in the project “Assets” folder
    - Call it “**gameOver”**



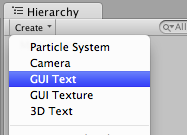
You should now see that you are editing scene **gameOver:**



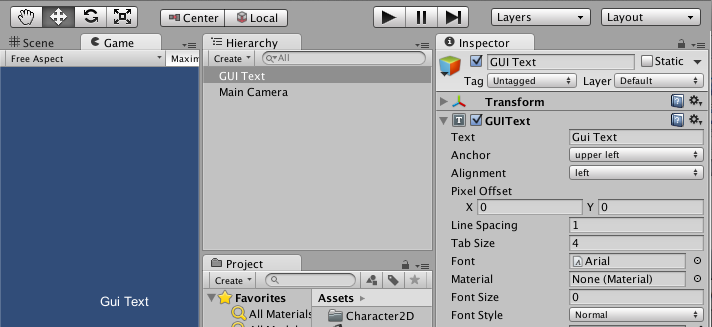
## Use GUIText to display the message “GAME OVER”

Add a big text message to the user using GUIText gameObjects:

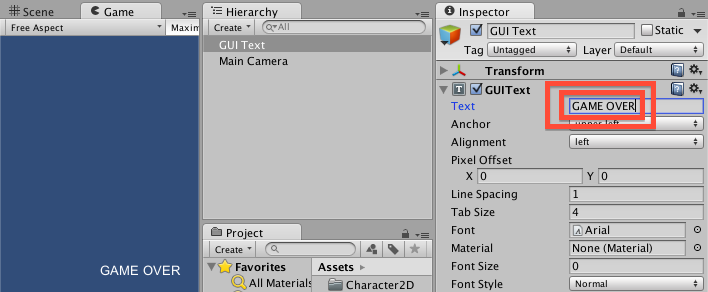
* From the “Create” menu of the **Hierarchy**, choose GUIText



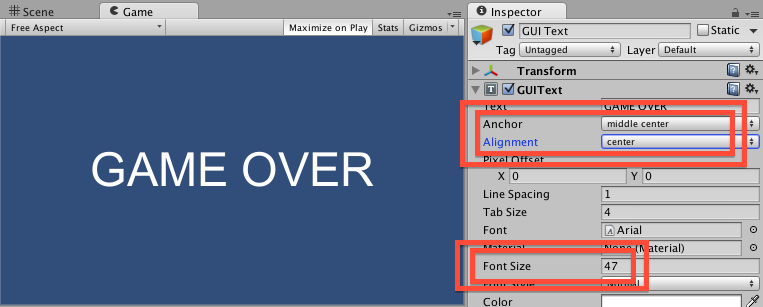
* You should now see a new GUIText gameObject in the **Hierarchy:**
  + **NOTE:** you can’t see the contents of a GUIText in the **Scene** panel, so click the **Game** panel, and you’ll see the preview of GUIText there with a blue background
  + **OR** just run your game scene – you’ll see just the text, since there is nothing else in our scene except for the camera …



* with the new **GUIText** gameObject selected in the **Hierarchy**
  + in the **Inspector** change its GUIText Text property to read “GAME OVER”



* with the new **GUIText** gameObject selected in the **Hierarchy**
  + in the **Inspector** change its GUIText properties as follows:
    - set Anchor to “middle center”
    - set Alignment to “center”
    - increase Font Size to 47
      * TIP: move mouse over the words Font Size and click and drag to the right to increase the number …
  + this shoud result in your GAME OVER message appearing as big text in the center of the **Game** panel

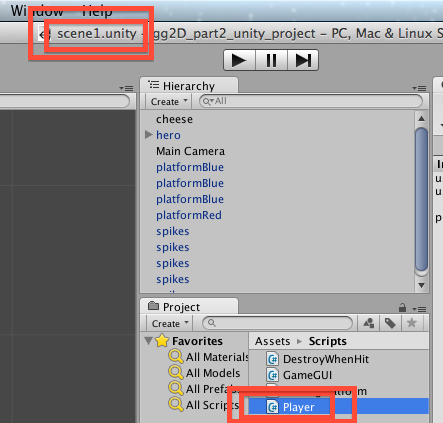


## Edit Player code, to load scene “gameOver” when lives less than zero

We now need to edit our Player script class, so that when the last life is lost (lives < 0) we make Unity change to our game over scene.

Let’s save our gameOver scene, go back to scene1, and then edit the Player script class …

* Save your **gameOver** scene
* Reload **scene1**
  + Double click **scene1 i**n the Assets folder of the **Project** panel
* Double click the **Player** script class file to load it into the **Monodevelop** editor



* Edit the **code as follows,** so that we tell the Unity **Application** to load the “**gameOver”** scene when lives are all lost:

*using UnityEngine;  
using System.Collections;  
  
public class Player : MonoBehaviour {  
    //  
    // public getters  
    //  
    public int GetScore(){  
        return score;  
    }  
      
    public int GetLives(){  
        return lives;  
    }  
  
    //  
    // properties  
    //   
    private int score = 0;  
    private int lives = 3;          
    private float DEATH\_Y = -5f;  
  
    //  
    // methods  
    //***private void Update(){  
        if(lives < 0){  
            Application.LoadLevel("gameOver");  
        }  
    }** *private void OnTriggerEnter2D(Collider2D c){  
        string tag = c.tag;  
  
        if("Food" == tag){  
            score++;  
            audio.Play();  
        }  
  
        if("Spikes" == tag){  
            lives--;  
            MoveToStartPosition();  
        }  
    }  
  
    private void MoveToStartPosition(){  
        Vector3 startPosition = new Vector3(0,5,0);  
        transform.position = startPosition;  
    }  
      
}*

## Playtest your game

Starting with the editor in **scene1**, run your game. Then keep falling off platforms to make the number of lives less than zero …. What happens?

The game did NOT load our **gameOver** level ?? Why not ?? And what is that error message:

Level 'gameOver' (-1) couldn't be loaded because it has not been added to the build settings.

To add a level to the build settings use the menu File->Build Settings...

UnityEngine.Application:LoadLevel(String)

Player:Update() (at Assets/Scripts/Player.cs:28)

This is what has happened …

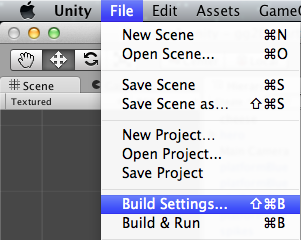
* Unity will only let you load a scene if the scene you are trying to load is included in the list of scenes that will be packaged up into the final ‘built’ application (whether a stand along desktop application, or web player game, or mobile phone app, or console game etc..)
* Sometimes you will create scenes just to TEST things out
  + You’ll never include these in your final build
* Unity does not want you to be able to write in code the loading of a scene that will cause an error when the final application is built for deployment or selling etc.

The SOLUTION …. You must now start telling Unity which scenes to include in your build.

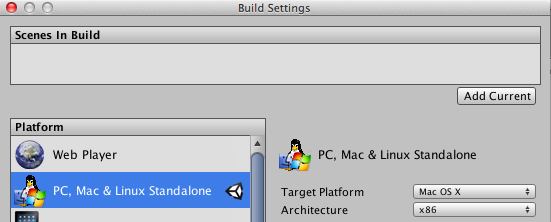
## Add ‘scene1’ and ‘gameOver’ scenes to your build list

Let’s add ‘scene1’ to the build:

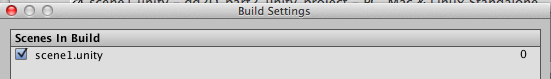
* When editing **scene1** choose menu: File | Build Settings …



* Delete any scenes that might be listed already, then click the button “Add Current”:



* You should now see **scene1** added to the list of Scenes in Build:



* Now load scene **gameOver**
  + You can either close the Build Setting dialog window, and re-open it after loading **gameOver**, or just move this window to one site
* With scene **gameOver** loaded, go back to the Build Settings diaglog, and click “Add Current” again
  + You should now see **scene1** and **gameOver** listed as scenes in the build
  + (note the scene list also have corresponding numbers, starting at 0 – Application.LoadLevel() also accepts the integer ‘index’ of the scene to load, as an alternative to the text name of the scene file …)

## Playtest your game

Close the Build Settings, and reload **scene1**, then run your game. Then keep falling off platforms to make the number of lives less than zero

You should now see the GAME OVER message once your lives gets below zero ☺

**Congratulations**

**You have now created part 2 of the tutorial !**