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

Gravity Guy 2D (2014) - a little computer game...

Part 3 – a bit more …



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

CONTENTS

1 Aims of this part of the tutorial 3

2 Create a prefab for ‘cheese’ objects 4

3 Create a folder to tidy up your scenes 5

4 Tidy up hierarchy – child items to empty container objects 6

5 Create rollover button, to navigate from Game Over scene back to level 1 playing 7

6 Set the ‘resolution’ of the application build to 800 x 600 pixels 10

7 Different sounds for different collisions 12

8 Congratulations … 14

# 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:

* **Create a prefab for ‘cheese’ objects**
  + **Containing the Collider2D and the “Food” tag**
* **Create folder to tidy up your scenes**
* **Tidy up hierarchy – child items to empty container objects**
* **Create rollover button, to navigate from Game Over scene back to level 1 playing**
* **Set the ‘resolution’ of the application build to 800 x 600 pixels**
* **Different sounds for different collisions**

# Create a prefab for ‘cheese’ objects

## Create a new empty prefab named ‘cheese’

Select the ‘prefabs’ folder of the **Project panel**, and create a new empty prefab named ‘cheese’.

## Copy all the components and properties of the cheese gameObject into your new prefab

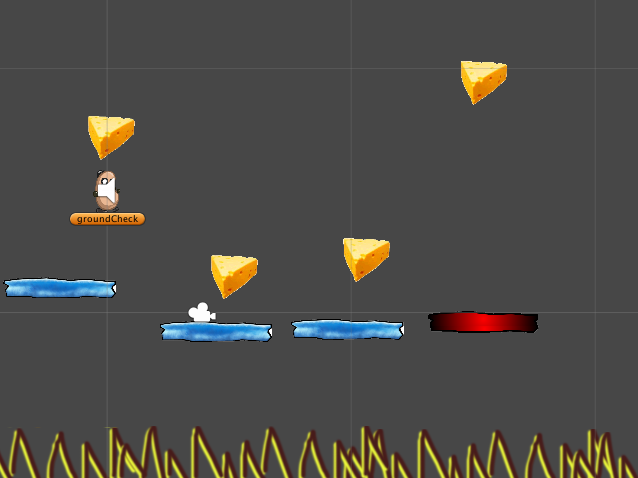
‘Populate’ your new cheese prefab, by dragging from the ‘cheese’ gameObject in the **Hierarchy** down into the new empty ‘cheese’ prefab.

You should now see the cheese prefab turn into a blue cube.

## Create some new cheese game objects from your cheese prefab

Add some new cheese gameObjects in the scene:

* Several times, drag from the cheese prefab in the **Project panel** into the **Scene panel**
* Each time you should be adding a new cheese gameObject in the current scene

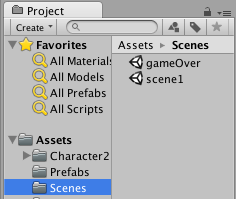


# Create a folder to tidy up your scenes

## Create folder ‘scenes’ and drag the 2 scenes there

Select the ‘Assets’ folder of the **Project panel**, and do the following:

* Create a new folder ‘Scenes’
* Drag ‘scene1’ and ‘gameOver’ into this Scenes folder

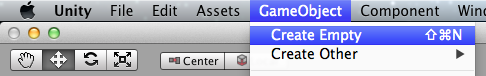


# Tidy up hierarchy – child items to empty container objects

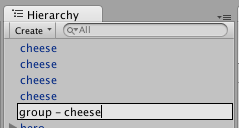
## Create ‘empty’ game object in Hierarchy, named ‘group – cheese’

Create a new empty gameObject in the Hiearchy:

* Select menu: GameObject | Create Empty

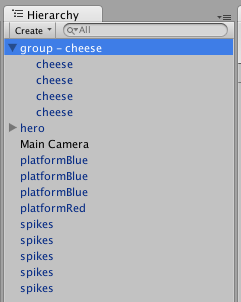


* Rename the new empty gameObject in the Hierarchy as ‘group – cheese’



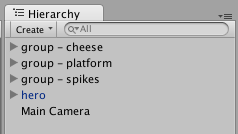
## In Hierarchy, drag all ‘cheese’ gameObjects into the new group object

In the **Hierarchy** drag each cheese gameObject into the ‘group – cheese’ object:



## Repeat with ‘group – spikes’ and ‘group – platform’

In the **Hierarchy** repeat this procedure to enchild all the spikes and platform gameObjects into new empty gameObjects:



# Create rollover button, to navigate from Game Over scene back to level 1 playing

## Add the 2 images and button script to your project

Locate the following files, and drag/add then to your **Project** Assets folder:

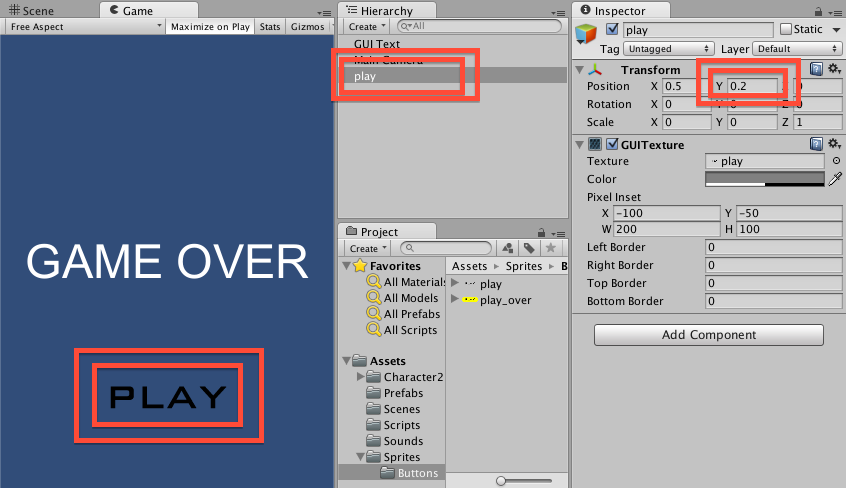
* RolloverButton.cs
  + The script we’ll add to any GUITexture image to make it a button
  + Put this into **Project panel** Assets/Scripts
* play.png
* play\_over.png
  + the 2 images for our button the make Unity go back to our game playing scene
  + Put these into **Project panel** Assets/Sprites/Buttons

## Turn the ‘play’ image into a GUITexture gameObject on the gameOver scene

Ensure you are editing the scene **gameOver** (double click the scene file to make Unity edit the scene).

Create a GUITexture gameObject,, named ‘play’, based on the ‘play’ image:

* FIRST: ensure the ‘play’ image is selected in the **Project** panel
* THEN: in the **Hierarchy** choose the Create ‘GUITexture’
* A new GUITexture gameObject should now have been created in your scene
  + Set the Y value of this object’s Transform – Position to 0.2
  + (to ensure this button is near the bottom of the screen, and not obscured by the GAME OVER message)
* You’ll need to view the **Game panel** to see this image object
  + Since GUIText and GUITexture objects ‘float’ above other scene objects in a special GUI layer at run time

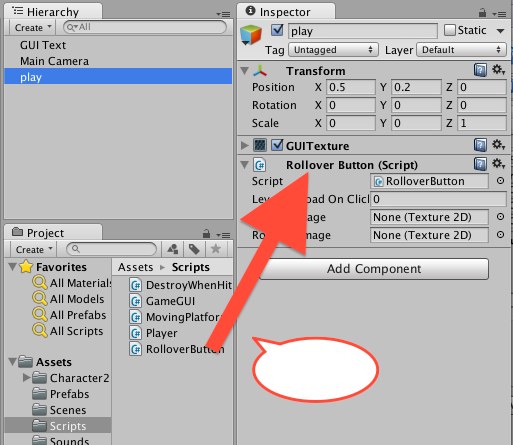


## Add the RolloverButton script to the ‘play’ GUITexture gameObject

To turn the GUITexture image into a rollover button, we need to add an instance of the RolloverButton script to our ‘play’ gameObject:

* Select the ‘play’ gameObject in the **Hierarchy**
* Drag the ‘RolloverButton’ script from the Scripts folder of the **Project panel** into the **Inspector** (or directly onto the ‘play’ gameObject in the **Hierarchy**)

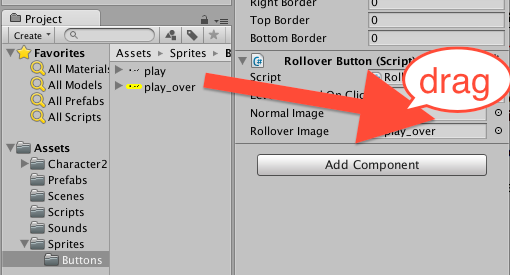
You should now see a RolloverButton script component in the **Inspector** components for the ‘play’ gameObject:



## Drag the ‘normal’ and ‘rollover’ button images into the script variables

To turn the GUITexture image into a rollover button, we need to add an instance of the RolloverButton script to our ‘play’ gameObject:

* Ensure the ‘play’ gameObject is selected in the **Hierarchy**
* In the **Project panel** select the folder containing the‘play’ and ‘play\_over’ images
* Drag the ‘play’ image in the ‘**Normal Image: None (Texture 2D)**’ variable slot in the **Inspector** for the RolloverButton script component
* Drag the ‘play\_over’ image in the ‘**Rollover Image: None (Texture 2D)**’ variable slot in the **Inspector** for the RolloverButton script component
* In this case, since we want the button click action to make Unity change to scene number 0 (**scene1**) we can leave the “**Level To Load On Click**” as 0
  + If the scene to load was a different number, then we’d replace zero with the correct scene number (which you can see on the right hand side of the Build Settings dialog window)



# Set the ‘resolution’ of the application build to 800 x 600 pixels

**WHAT are we making?**

Unity projects can be ‘built’ into different kinds of computer program:

* Stand along PC, Apple, Linux desktop applications
* Web-player games
* Games console games
* Mobile phone ‘apps’ for iOS, Android, Windows phone etc.

Once we know the destination ‘device’ that our game will be ‘deployed’ onto, we can set the **Game** panel dimensions to the device **width** and **height**

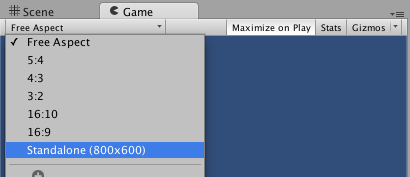
Even if we are creating a ‘proof of concept’ it is important to FIX THE RESOLUTION of our game, so we can design the screen layout, and find / create 2D graphic assets of the correct size

As a rule of thumb, If in doubt, build to 800 x 600 pixels (width x height)

## If listed, choose the desired Game resolution from Game panel drop down

If the build settings have been set up correctly, or they are inherited since we started our game on a pre-ade Unity project (like Gravity Guy), then the **Game panel** resolution can be set very simply:

* Choose the screen resolution (800 x 600) from the drop down menu at the top of the **Game panel**



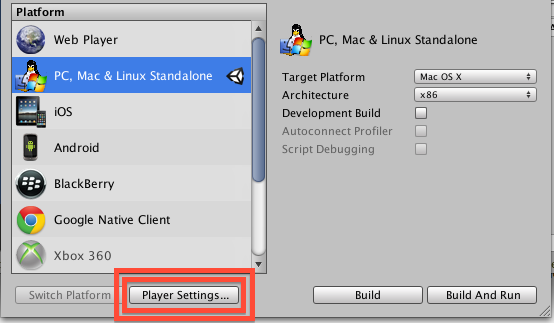
Once chosen, you’ll see that the **Game panel** now always scales its display with the correct width/height proportion:



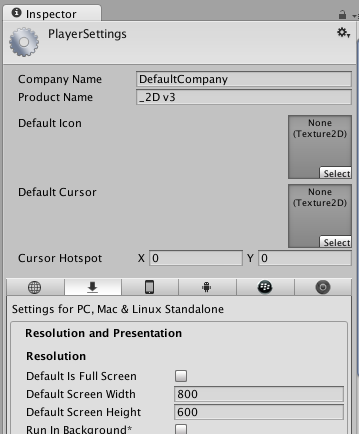
## If desired resolution not listed, set it via the Build settings option

To change/set a resolution to have added to the Game panel menu, do the following:

* Open the Build settings dialog, by choosing menu: **File | Build Settings …**
* Click the button ‘**Player Settings …**”



* You can set the default screen width and height in the **Inspector**, which is now showing the Player Settings properties:



# Different sounds for different collisions

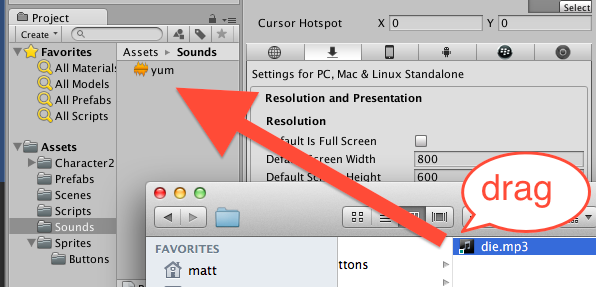
## Add new audio sound clip file to your project

Most games play one of several different sounds, depending on what even has occurred. We are going to enhance our game to do the following:

* If “Food” hit, play the “yum” audio sound clip
* If “Spikes” hit, play the “die” audio sound clip

First, add the “die” audio sound clip to our project:

* Drag the “die.mp3” file into the **Project panel** folder “Sounds”



## Change our “Player” script class to play different sounds when appropriate

Once we are dealing with more than one audio clip, our script needs a variable for each sound clip file.

So first add public variables for the 2 sound clips to our Player script class:

* 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 variables “yumSound” and “dieSound”

*using UnityEngine;*

*using System.Collections;*

*public class Player : MonoBehaviour*

*{*

**public AudioClip yumSound;**

**public AudioClip dieSound;**

*public int GetScore()*

*{*

*return score;*

*}*

**One reason to make a variable ‘public’ – can be set in the Inspector**

Once an instance of a script class has been added as a component of a gameObject, the public variables for that instace can be set in the **Inspector**

* Numeric variables can just be typed in
* Media objects (images, sounds, 3D models, materials, prefabs) can be drag-and-droppped
* Drag-and-drop can also be used when variables are gameObjects or script instances as well

This makes linking different parts of our game, and our media files, very straightforward and natural – once you’ve got the hand of it …

Note – rather annoyingly, in the Inspector the first letter of variables are capitalised ….

Next we are going to change our code. Rather than sending a “Play()” message to the “audio” component, we need to send a “PlayerOneShot” messages AND pass as a parameter the sound clip variable, so the Audio Source component knows which sound clip to play:

Change method OnTriggerEnter2D so that it looks as follows:

*private void OnTriggerEnter2D(Collider2D c)*

*{*

*string tag = c.tag;*

*if("Food" == tag)*

*{*

*score++;*

**audio.PlayOneShot(yumSound);**

*}*

*if("Spikes" == tag)*

*{*

*lives--;*

**audio.PlayOneShot(dieSound);**

*MoveToStartPosition();*

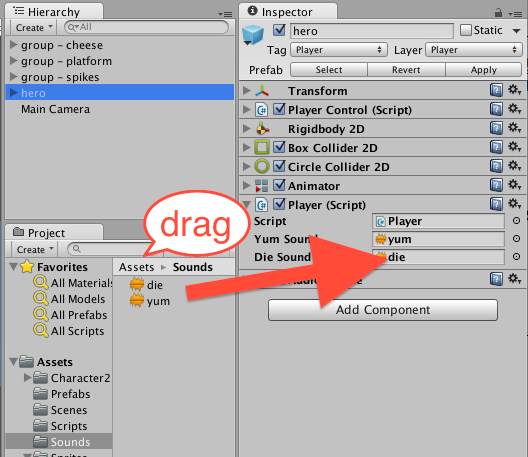
*}*

*}*

## Associate the audio sound clip files with our yumSound and dieSound variables

Now we need to link the sound clip files in the **Project panel** with the yumSound and dieSound public variables in the instance of our Player script in our ‘hero’ guy in scene1:

* Ensue you have SAVED your Player script changes
  + And fix any erorrs !
* Ensure you are editing scene “**scene1”**
* Select the ‘hero’ gameObject in the **Hierarchy**
* Select the “Sounds” folder in the **Project panel**
* You can now drag-and-drop the “yum” sound from the Sounds folder over the “Yum Sound” variable in the Player script component in the Inspector
  + And do the same with the “die” sound clip for variable “Die Sound”



## Playtest your game

When player hits spikes, the ‘die’ sounds should play, when player hits cheese, the ‘yum’ sound should play – simples!

# Congratulations …

**Congratulations**

**You have now completed Gravity Guy part 3 !**