|  |  |
| --- | --- |
| 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 9 – timers and timing …



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 2

2 Improve death – temporary invulnerability ! 3

3 Show the user – display a ‘shield’ around hero when invulnerable! 5

4 Simple timers 7

5 FULL LISTINGS 11

# 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:

* Temporary invulnerability – cannot die!
* Display visible shield around hero when invulnerable
* Adding a countdown timer

# Improve death – temporary invulnerability !

## Multiple lost life problem

One problem with the combination of **frame-based methods** (e.g. Update(), FixedUpdate()) and e**vent-based methods** (OnTriggerEnter2D(), OnGUI()), and the Unity **physics** system, is that sometimes things happen in odd or unplanned combinations.

This can lead, for example, to losing multiple lives, because a character is not moved back to the start position before hitting a second object etc.

One robust solution, is to ensure certain actions cannot happen again within a certain ‘safety’ time period, e.g. 0.5 seconds. In some games this temporary invulnerability is visually shown to the user (e.g. when you get a new space ship in Asteroids, for a second or two you see a force field circle around the new ship, and you won’t become vulnerable again until there is a clear area around you.

Many times in computer games, a simple solution is to compare the current time in the game (**Time.time**) with the next time something is allowed to happen (e.g. **nextTimeAllowedToDie**). When an event happens, e.g. we lose a life, we also reset the next time for that event to happen to be the current time plus our delay, e.g.:

nextTimeAllowedToDie = Time.time + delayBetweenDeaths;

So lets add this feature to our game, so we stop that annoying, losing 2 lives when you hit the spikes problem …

**NOTE: Time.time**

In Unity our game timer counts the number of seconds (float) since the application started running. So after 1-and-a-half seconds, **Time.time** will be 1.5 and so on ….

## Declare variables for nextTimeAllowedToDie and delayBetweenDeaths

Add the following variables to your **Player.cs** class:

**public float delayBetweenDeaths = 5;**

**private float nextTimeAllowedToDie = 0;**

Note – we initialise it to zero, so from as soon as the game starts (**Time.time** starts at zero) we are allowed to lose a life.

We have set the delay between deaths to a large time (5 seconds), so you can test it easily – die ones, see lives reduce by 1, and then die again quickly, and you should see you don’t lose another life yet.

## Refine our LoseLife() method to only lose life is it’s time to die …

Modify method **LoseLife()** method in **Player.cs** class to the following:

private void LoseLife()

{

if(Time.time > nextTimeAllowedToDie){

lives--;

// update next time allowed to die for a future time ...

nextTimeAllowedToDie = Time.time + delayBetweenDeaths;

}

if(lives < 0)

{

Application.LoadLevel("scene1\_GameOver");

}

playerDisplay.UpdateLivesImage(lives);

MoveToStartPosition();

audio.PlayOneShot(dieSound);

}

We have enclosed an IF-statement around our statement that decrements (reduces by 1) the number of lives. So we only reduce lives if the current time has passed our next time to be allowed to die. If we do remove a life, we also then reset the next time to die to be the current time PLUS our delay between lives – so we get some more seconds or fractions of seconds of invulnerability before we can lose another life.

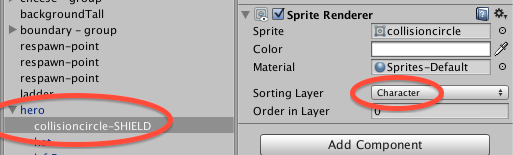
# Show the user – display a ‘shield’ around hero when invulnerable!

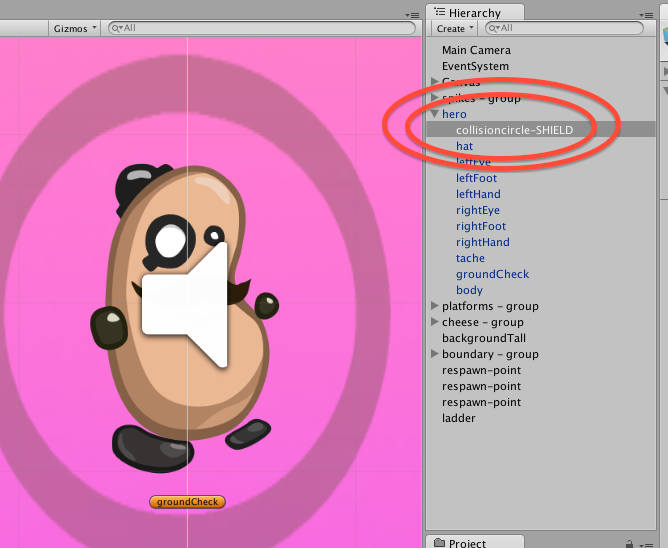
## Add a shield Sprite as a child inside hero

Let’s add a shield oval around hero. We’ll add it as a ‘child’ object of hero in the Hierarchy, so when hero moves, it moves with him!

Do the following:

* Drag image ‘collisioncircle’ from the provided assets into the Sprites folder of your Project panel
* Drag collisioncircle from Sprites onto the Scene, renaming it **collisioncircle-SHIELD**
  + Set its **Sorting Layer** to **Character** (so we can see it in front of the Background)
  + and position / resize so it is around our hero character
* now ‘child’ **collisioncircle-SHIELD** into the hero gameObject in the Hierarchy



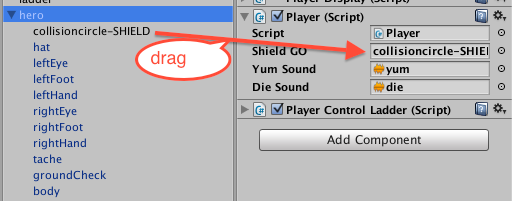


## Modify Player.cs to show shield when invulunerable

Our **Player** object needs a reference to this shield gameObject – let’s do this through a public GameObject variable, which we can set via the inspector:

public GameObject shieldGO;

Select hero in the Hierarchy , expand its children and drag the **collisioncircle-SHIELD** gameObject into the slot for **Shield GO** in the Player (Script) component:



Next we’ll write a method that will show the shield when the time to die has not been reached (i.e. we are invulnerable), and hide the shield if we can die (current time PAST the next time to die). The **SetActive()** method disables gameObjects if set to false:

private void DisplayShieldIfCannotDie()

{

bool cannotDie = (Time.time <= nextTimeAllowedToDie);

// show shield if cannot die

// else hide it

shieldGO.SetActive( cannotDie );

}

Finally, we’ll add a call to our new method EVERY FRAME in the Update() method:

void Update()

{

*CheckGameWon();*

*CheckDeathYReached();*

DisplayShieldIfCannotDie();

}

## Playtest your game

After you lose a life, for a few seconds () you’ll see a ‘shield’ of invulnerability around our hero guy, during which time he cannot lose a life – fantastic!

# Simple timers

## In many games there is some form of countdown or countup timer …

Examples of timers in games include:

* Lose life if level not complete before countdown timer gets to zero
* Pick up a shield of invulnerability, that lasts for 2 seconds
* Drop a bomb that will explode in 5 seconds
* Survive as long as possible during a bonus mini-level, and for each 10 seconds you survive you get an extra life / more money etc.
* And so on…

## The CountdownTimer.cs script

Add an instance of the provided **CountdownTimer.cs** script to the Main Camera:

* Select the **Main Camera** in the **Hierarchy**
* Drag a copy of the **CountdownTime** script to the **Inspector** (or directly onto the **Main Camera**)

NOTE: just as with ButtonActions, if we need to have a runtime object of a class, and there isn’t a natural home (such as in the player’s character, or in a pickup object like a key), then the Main Camera is one handly place to add instances of classes to keep things tidy.

## The GameManager class – overview

We shall create a Game Manager class to handle the following responsibilities:

* Reset the timer at the beginning of the level
* Load the Game Over screen, if the timer goes below zero – checking every frame in method Update()
* Update the remaining seconds display in a UI Text object, again every frame from method Update()

## The GameManager class – reset timer when scene begins

Create a new C# script named **GameManager**. When the scene starts, we need to get a reference to our CountdownTimer object (also in GameManager), and reset it to the number of seconds for this level (let’s give the player 20 seconds for now, but if we make this public then we can change it later in the **Inspector** if we want to). We need a private variable to be a reference to our **CountdownTimer** object:

public int timeForLevel = 20;

private CountdownTimer countdownTimer;

void Start() {

countdownTimer = GetComponent<CountdownTimer>();

countdownTimer.ResetTimer(timeForLevel);

}

## The GameManager class – check each frame for game over condition (timer < 0)

Each frame we test the game over conditions, in this case if the seconds remaining in our timer go below zero. If this is the case it’s time to load the Game Over scene. Since we are going to also be wanting to update the timer display every frame, we’ll get the seconds left, and then call our game over test (in its own method) passing the remaining seconds, doing this every frame from Update():

void Update()

{

int secondsLeft = countdownTimer.GetSecondsRemaining();

CheckGameOver(secondsLeft);

}

private void CheckGameOver(int secondsLeft)

{

if(secondsLeft < 0)

{

Application.LoadLevel("scene1\_GameOver");

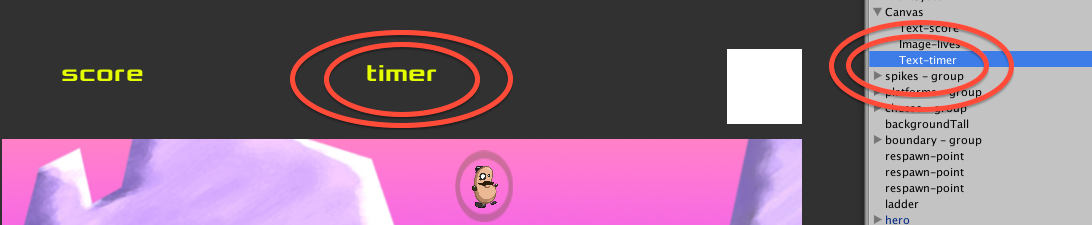
}

}

## The GameManager class – update display of seconds left to user (every frame)

Every frame let’s update the remaining seconds display in a UI Text object.

First, let’s create a new UI Text object to display the seconds remaining. The simplest way is to duplicate the **Text-score** UI text object, rename the copy **Text-timer**, position it to be at TOP – CENTER, and change its default text to something meaningful like ‘timer’ so we know which UI Text it is in the **Game** panel:



Next our **GameManager** class needs a public Text object reference variable, into which we can drag our Text-timer UI object:

using UnityEngine;

using System.Collections;

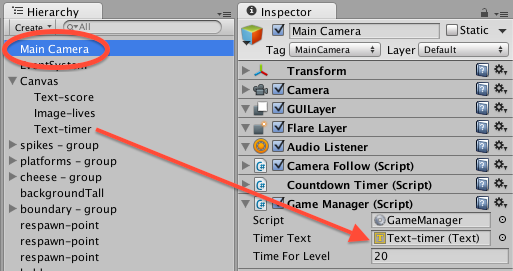
using UnityEngine.UI;

public class GameManager : MonoBehaviour

{

// public

public Text timerText;



Each frame, from method **Update()** we wish to call method **UpdateTimerDisplay**():

private void Update()

{

int secondsLeft = myTimer.GetSecondsRemaining();

CheckGameOver(secondsLeft);

UpdateTimerDisplay(secondsLeft);

}

private void UpdateTimerDisplay(int secondsLeft)

{

string timerMessage = "Time left = " + secondsLeft;

timerText.text = timerMessage;

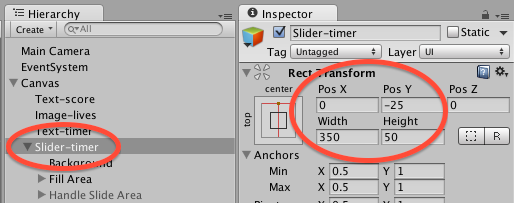
}

# Using a UI Slider as a ‘progress bar’ to show timer status

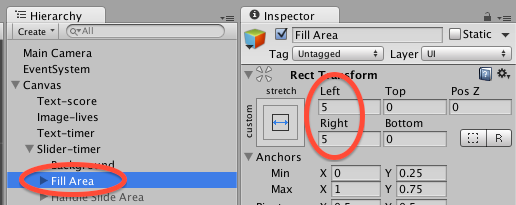
## Create graphical UI Slider ‘bar’ to show proportion of timer remaining

Create our timer bar on screen as follows:

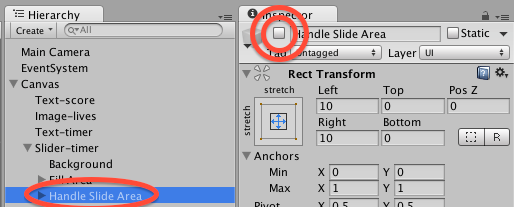
* Select the Canvas in the Hierarchy and add a UI Slider, name it Slider-timeLeft
* Position it TOP – CENTER
* Set width to 350 and height to 50, and Pos Y to -25 (so it is now below the Text-timer object)



* Click the triangle to show its children, and de-activate the child **Handle Slide Area**
  + You’ll see the ‘circle’ disappear (since the user will not be dragging it, since this is a display-only slider)



* Select the Background child
  + Set its color to red (instead of White)
* Select the Fill Area child
  + Set its Right value to 5 (instead of 15)



## Add GameManager.UpdateTimerSlider()

Let’s add a new public variable so our GameManager object has a reference to the UI Slider-timer:

public Slider timerSlider;

Add a new method to update the ‘value’ property of the slider, to a value between 0.0 and 1.0 representing the proportion of time remaining (we can use the handy pre-written CountdownTimer.GetProportionTimeRemaining() method to return exactly such a value to us:

private void UpdateTimerSlider()

{

float proportionRemaining = countdownTimer.GetProportionTimeRemaining();

timerSlider.value = proportionRemaining;

}

Now we need to simply call our new method every frame from Update():

*void Update()*

*{*

*int secondsLeft = countdownTimer.GetSecondsRemaining();*

*CheckGameOver(secondsLeft);*

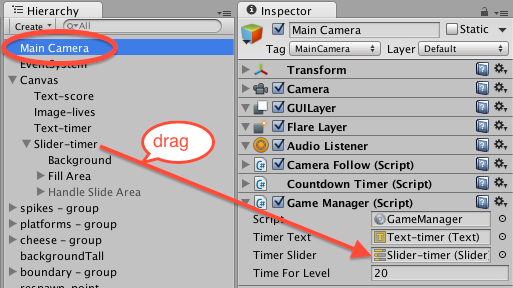
*UpdateTimerDisplay(secondsLeft);*

UpdateTimerSlider();

*}*

Finally, we need to drag the UI Slide-timer in the Hierarchy into the public variable slot of the MainCamera GameManager (Script) component, so the UI Slider value can be updated at run-time:

* Select the Main Camera in the Hierarchy
* Drag Slider-timer into the Timer Slider variable slot in the Game Manager (Script) component in the Inspector:



### Playtest your game

Start on the Welcome scene and click the button to play the game.

Now you have some timer after losing a life when you won’t lose another – displayed via a ‘shield’ around your hero character.

Also we have a simple countdown timer (whose value is shown both in text and as a progress bar), which when it gets below zero means you have lost the game.

**Congratulations**

**You have now created part 9 of the tutorial !**

# FULL LISTINGS

## Player

using UnityEngine;

using System.Collections;

public class Player : MonoBehaviour {

public GameObject shieldGO;

public AudioClip yumSound;

public AudioClip dieSound;

private float delayBetweenDeaths = 5f;

private float nextTimeAllowedToDie = 0;

private PlayerDisplay playerDisplay;

private int lives = 3;

private int score = 0;

private float deathY = -15;

//-----------------------------

void Start()

{

playerDisplay = GetComponent<PlayerDisplay>();

playerDisplay.UpdateScoreText(score);

playerDisplay.UpdateLivesImage(lives);

}

//-----------------------------

void Update()

{

CheckGameWon();

CheckDeathYReached();

DisplayShieldIfCannotDie();

}

//---------------------------------

private void DisplayShieldIfCannotDie()

{

bool cannotDie = (Time.time <= nextTimeAllowedToDie);

// show shield if cannot die

// else hide it

shieldGO.SetActive( cannotDie );

}

//-----------------------------

private void CheckDeathYReached()

{

float y = transform.position.y;

if(y < deathY){

LoseLife();

}

}

//-----------------------------

private void LoseLife()

{

if(Time.time > nextTimeAllowedToDie){

lives--;

// update our next time allowed to die for a future time ...

nextTimeAllowedToDie = Time.time + delayBetweenDeaths;

}

if(lives < 0)

{

Application.LoadLevel("scene1\_GameOver");

}

playerDisplay.UpdateLivesImage(lives);

MoveToStartPosition();

audio.PlayOneShot(dieSound);

}

//-----------------------------

private void MoveToStartPosition()

{

GameObject respawnGO = ChooseRandomObjectWithTag("Respawn");

Vector3 startPosition = respawnGO.transform.position;

transform.position = startPosition;

// remove all horizontal/vertical movement when respawned

rigidbody2D.velocity = Vector2.zero;

}

//-----------------------------

private GameObject ChooseRandomObjectWithTag(string tag)

{

GameObject[] taggedObjects = GameObject.FindGameObjectsWithTag(tag);

int numTaggedObjects = taggedObjects.Length;

int randomIndex = Random.Range(0, numTaggedObjects);

return taggedObjects[randomIndex];

}

//-----------------------------

void OnTriggerEnter2D(Collider2D hit)

{

if(hit.CompareTag("Food"))

{

score++;

playerDisplay.UpdateScoreText(score);

Destroy (hit.gameObject);

audio.PlayOneShot(yumSound);

}

if(hit.CompareTag("Spikes"))

{

LoseLife();

}

}

//------------------------------

private int CountObjectsWithTag(string tag)

{

GameObject[] foodObjects = GameObject.FindGameObjectsWithTag(tag);

return foodObjects.Length;

}

//-------------------------------

private void CheckGameWon()

{

print ("hello");

int numFoodObjects = CountObjectsWithTag("Food");

print ("number of food objects left = " + numFoodObjects);

if(numFoodObjects < 1)

{

Application.LoadLevel("scene3\_GameWon");

}

}

}

## CountdownTimer.cs

using UnityEngine;

using System.Collections;

public class CountdownTimer : MonoBehaviour

{

private float countdownTimerStartTime;

private int countdownTimerDuration;

//-----------------------------

public int GetTotalSeconds()

{

return countdownTimerDuration;

}

//-----------------------------

public void ResetTimer(int seconds)

{

countdownTimerStartTime = Time.time;

countdownTimerDuration = seconds;

}

//-----------------------------

public int GetSecondsRemaining()

{

int elapsedSeconds = (int)(Time.time - countdownTimerStartTime);

int secondsLeft = (countdownTimerDuration - elapsedSeconds);

return secondsLeft;

}

//-----------------------------

public float GetProportionTimeRemaining()

{

float proportionLeft = (float)GetSecondsRemaining() / (float)GetTotalSeconds();

return proportionLeft;

}

}

## GameManager.cs

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class GameManager : MonoBehaviour

{

// public

public Text timerText;

public Slider timerSlider;

public int timeForLevel = 20;

// private

private CountdownTimer countdownTimer;

//--------------------------------------

void Start()

{

countdownTimer = GetComponent<CountdownTimer>();

countdownTimer.ResetTimer(timeForLevel);

}

//--------------------------------------

void Update()

{

int secondsLeft = countdownTimer.GetSecondsRemaining();

CheckGameOver(secondsLeft);

UpdateTimerDisplay(secondsLeft);

UpdateTimerSlider();

}

//---------------------------

private void UpdateTimerSlider()

{

float proportionRemaining = countdownTimer.GetProportionTimeRemaining();

timerSlider.value = proportionRemaining;

}

//--------------------------------------

private void CheckGameOver(int secondsLeft)

{

// GAME OVER if seconds < 0 !!!!!

if(secondsLeft < 0)

{

Application.LoadLevel("scene1\_GameOver");

}

}

//---------------------------

private void UpdateTimerDisplay(int secondsLeft)

{

string timerMessage = "Time left = " + secondsLeft;

timerText.text = timerMessage;

}

}