

Introduction to Unity

Step 3: introduce some „activities” to the game world with scripts

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

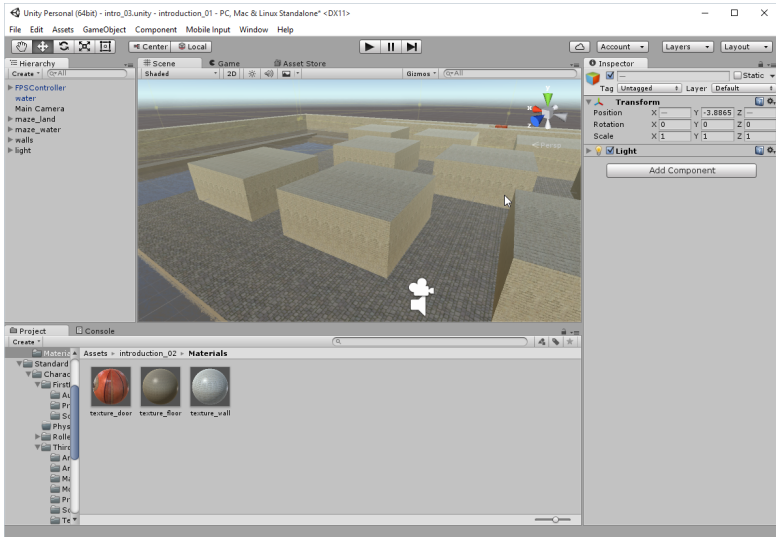
November 17, 2016

Table of contents

Scripts make it possible to add logic and more interactivity to our games, as well as customize interaction based on the players' actions.

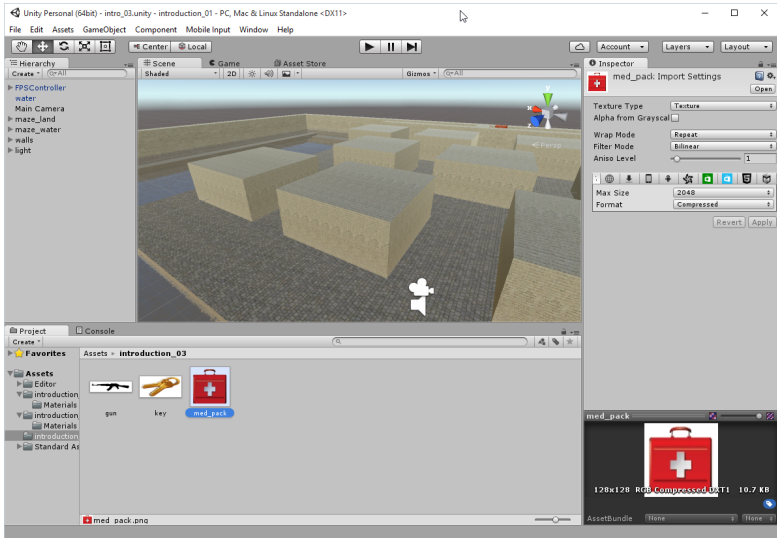
As we have mentioned in a first part of this tutorial, we can create scripts using both JavaScript and C#. While JavaScript is usually considered an easy and accessible scripting language, C# is usually favored by intermediate and advanced programmers, as it facilitates the programming workflow and makes it possible to develop more complex programs.

- 1 Open the previous project (scene intro_02).
- 2 Duplicate the scene we have been working on so far by saving it as intro_03 **File | Save Scene As**.
- 3 Create a container for all objects we created for the game world in the previous step:
 - 1 Create an empty object **Game Object | Create Empty** and change its name to `maze_land`.
 - 2 Change its position to $(x=0, y=0, z=0)$.
 - 3 In the **Hierarchy** window, select all objects which creates land maze (`maze_land`), and drag-and-drop these objects on the object labeled `maze`.
- 4 With the same method create containers for other object if you need it.



- 1 Create a new folder called `introduction_03`, inside the Assets folder.
- 2 Find / prepare texture for medical package (we will call this: medpack) object.
- 3 Find / prepare texture for key object.
- 4 Find / prepare texture for gun object.
- 5 Find / prepare sound for collecting things (for example: <http://soundbible.com/2084-Glass-Ping.html>).
- 6 Find the font you want to use as user interface font (for example: www.dafont.com, *Techno / LCD / Open 24 Display ST by Southype*) and download the font and unzip it (you should see the *.ttf file).

- 1 Select the current project folder `introduction_03`.
- 2 Select **Assets | Import New Asset**.
- 3 Browse to the folder where we downloaded the textures.
- 4 Select one of the texture saved before and click on **Import**.
- 5 Repeat above steps for the rest of textures.

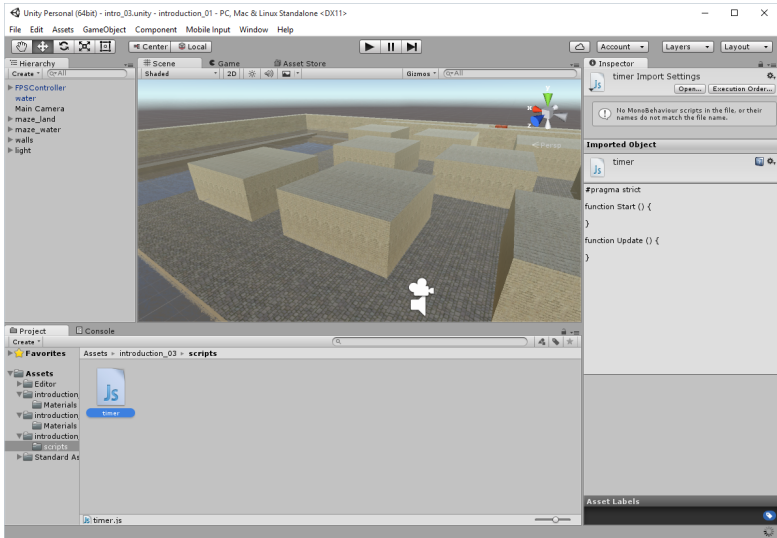


Create folder for the scripts

- 1 Select the folder introduction_03 and from the **Project** window, select **Create | Folder**.
- 2 Rename this folder scripts.

Create a new script

- 1 Check if the folder `scripts` is selected.
- 2 From the top menu, select **Assets | Create | JavaScript**.
- 3 Doing so should create a new JavaScript script within the folder labeled `scripts`.
- 4 Rename this script `timer`.



- 1 When the script has been created, we can see its content in the **Inspector** window.
- 2 Double-click on the script labeled `timer`. Doing so should open the default editor for Unity3D scripts: MonoDevelop¹.
- 3 Once in MonoDevelop, we can see that there are two functions created in the timer script by default:
 - The `Start` function is called when the script is first called. For example, if this script is linked to an object, this function will be called when the object is created or added to the scene.
 - The `Update` function is called every frame (that is, when the screen is refreshed).
- 4 Modify the timer script as follows (see next slide):

¹We could have changed Unity3D's preferences accordingly in **Edit | Preferences | External Tool**.

```
private var time: float;
```

```
function Start ()  
{  
}
```

```
function Update ()  
{  
    time++;  
    print(time);  
}
```

Assembly-UnityScript - introduction_03\scripts\timer.js - MonoDevelop-Unity

File Edit View Search Project Build Run Version Control Tools Window Help

MonoDevelop-Unity Press 'Control+' to search

Solution

- Introduction_01
 - Assembly-CSharp-Editor
 - Assembly-CSharp-firstpass**
 - Assembly-UnityScript
 - References
 - introduction_03
 - scripts
 - timer.js**
 - Standard Assets
 - New Unity Project

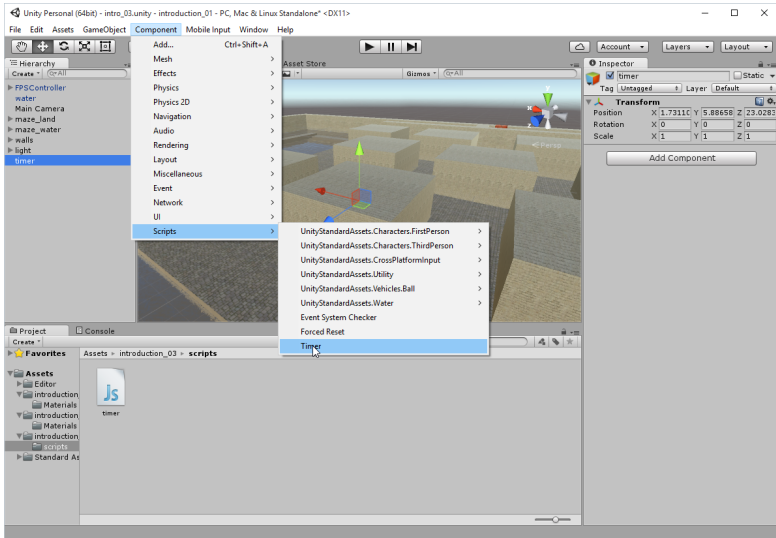
timer.js

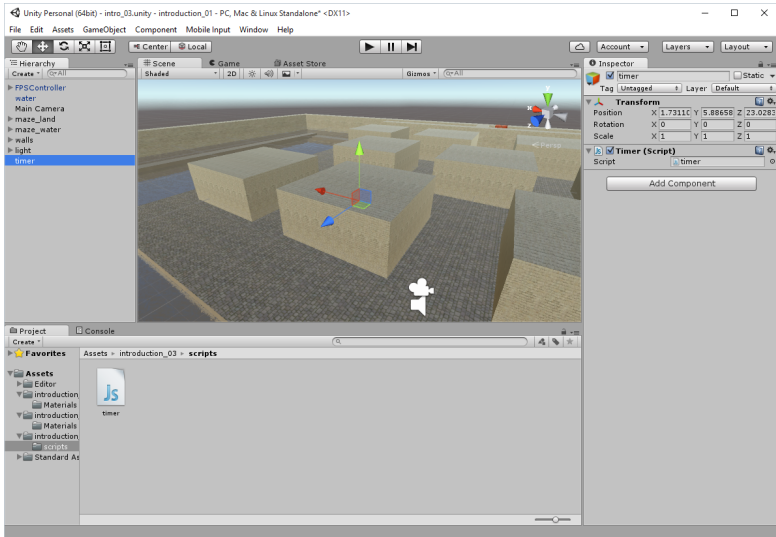
```
1 #pragma strict
2
3 private var time: float;
4
5 function Start () {
6
7 }
8
9 function Update () {
10     time++;
11     print(time);
12 }
```

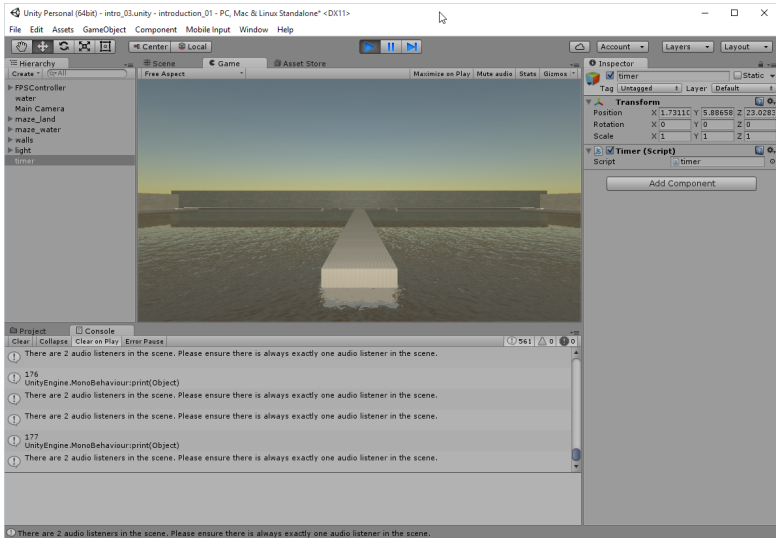
Errors Tasks

Attached script to an object

- 1 Create an empty object **Game Object | Empty Object** and rename it `timer`.
- 2 Attach the script by either
 - dragging-and-dropping the script from the `scripts` folder to the `timer` object, or
 - by selecting the object labeled `timer` and selecting **Component | Scripts | timer** from the top menu.
- 3 Now, if we click once on the `timer` object, the **Inspector** will reveal an additional component for this object, a script labeled `timer`.
- 4 Open the Console window (**Ctrl + Shift + C**) and play the scene (**Ctrl + P**). We can see that the counter is displayed and that its value increases over time.







Change script to use seconds not frames

We will use a built-in variable called `Time.deltaTime` to deal with seconds not frames. What a **deltaTime** is should be clear as we have talked about this in lecture *Game loop and time*.

- 1 Modify the script

```
function Update ()
{
    time = time + Time.deltaTime;
    print(time);
}
```

- 2 Play the scene.

Change script to display minutes and seconds

- 1 Modify the script

```
private var time: float;  
private var minutes: int;  
private var seconds: int;
```

```
function Start ()  
{  
}
```

```
function Update ()  
{  
    time = time + Time.deltaTime;  
    minutes = time/60;  
    seconds = time%60;  
    print(minutes + ":" + seconds);  
}
```

- 2 Play the scene.

Assembly-UnityScript - introduction_03\scripts\timer.js - MonoDevelop-Unity

File Edit View Search Project Build Run Version Control Tools Window Help

MonoDevelop-Unity

Press 'Control+' to search

Solution

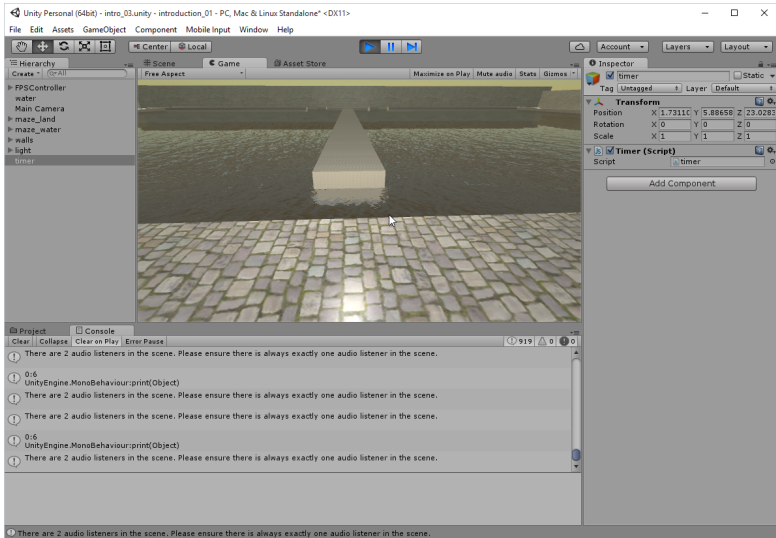
- Introduction_01
 - Assembly-CSharp-Editor
 - Assembly-CSharp-firstpass**
 - Assembly-UnityScript
 - References
 - introduction_03
 - scripts
 - timer.js**
 - Standard Assets
 - New Unity Project

timer.js

timer > Update

```
1 #pragma strict
2
3 private var time: float;
4 private var minutes: int;
5 private var seconds: int;
6
7 function Start () {
8
9 }
10
11 function Update () {
12     time = time + Time.deltaTime;
13     minutes = time/60;
14     seconds = time%60;
15     print(minutes + ":" + seconds);
16 }
```

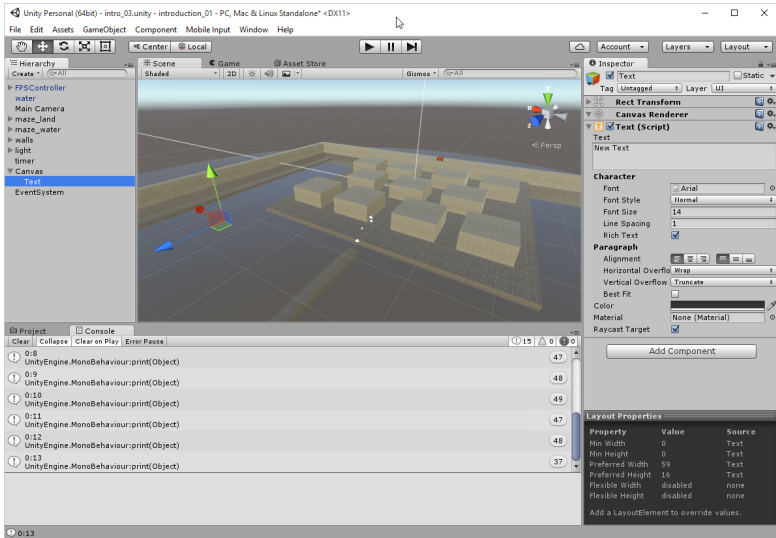
Errors Tasks



Displays the time on the screen

Create a **UI.Text** object

- 1 Create a **UI.Text** object: select **GameObject | UI | Text**. This should add **Canvas** with **Text** subelement and **EventSystem** object to our project (see **Hierarchy** window).
- 2 Rename **Text** object **UI_timer**.
- 3 Select **UI_timer** object and in **Inspector** window expand **Rect Transform** and select as a reference point **middle-center** and set (Pos X = 0, Pos Y = 0, Pos Z = 0).
- 4 If we switch to the game view, we should now see the default text *New Text* in the middle of the screen.
- 5 Change position so that *New Text* would be displayed at the bottom-left corner and use bigger font ().



Unity Personal (64bit) - intro_03.unity - introduction_01 - PC, Mac & Linux Standalone* <DX11>

File Edit Assets GameObject Component Mobile Input Window Help

Center Local

Scene Game Asset Store

Free Aspect Maximize on Play Mute audio Stats Gizmos

Inspector Account Layers Layout

Hierarchy Create All

- FPSController
- water
- Main Camera
- maze_land
- maze_water
- walls
- light
- timer
- Canvas
 - UT_timer
 - EventSystem

New Text

Project Console

Clear Collapse Clear on Play Error Pause

0:8 UnityEngine.MonoBehaviour:print(Object) 47

0:9 UnityEngine.MonoBehaviour:print(Object) 48

0:10 UnityEngine.MonoBehaviour:print(Object) 49

0:11 UnityEngine.MonoBehaviour:print(Object) 47

0:12 UnityEngine.MonoBehaviour:print(Object) 48

0:13 UnityEngine.MonoBehaviour:print(Object) 37

0:13

Inspector

Width 0 Height 0

200 25

Anchors

Pivot X:0 Y:0

Rotation X:0 Y:0 Z:0

Scale X:1 Y:1 Z:1

Canvas Renderer

Text (Script)

New Text

Character

Font Arial

Font Style Normal

Font Size 20

Line Spacing 1

Rich Text

Paragraph

Alignment

Horizontal Overflow Wrap

Vertical Overflow Truncate

Best Fit

Color

Material None (Material)

Raycast Target

Add Component

Layout Properties

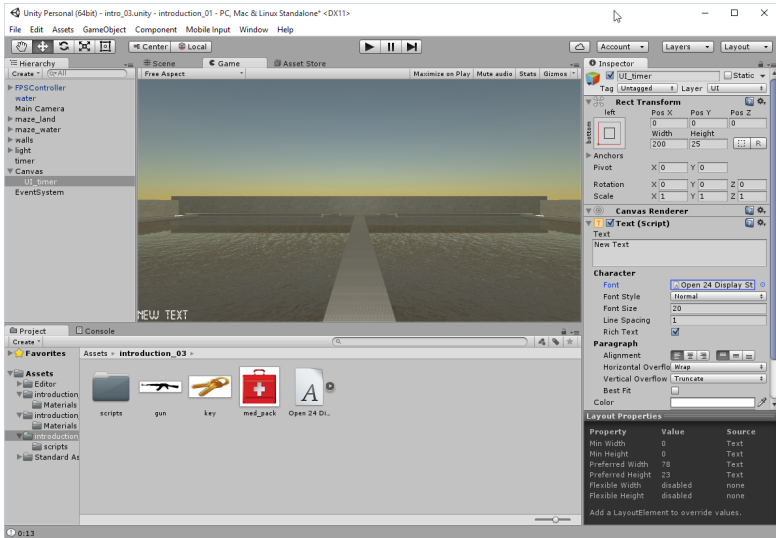
Property	Value	Source
Min Width	0	Text
Min Height	0	Text
Preferred Width	0.2	Text
Preferred Height	23	Text
Flexible Width	disabled	none
Flexible Height	disabled	none

Add a LayoutElement to override values.

Displays the time on the screen

Changing the font

- 1 Select the folder `introduction_03`, and import the font you have prepared at the beginning into Unity3D (**Assets | Import New Asset**).
- 2 Select the **UI.Text** object labeled `UI_timer`.
- 3 In the **Inspector** window, click on the small circle to the right of the label **Font**.
- 4 This should open a window labeled **Select font** that includes the new font you have just downloaded. Select this font and close the font selection window. The new font should now appear in the **Font** property of the previous object.

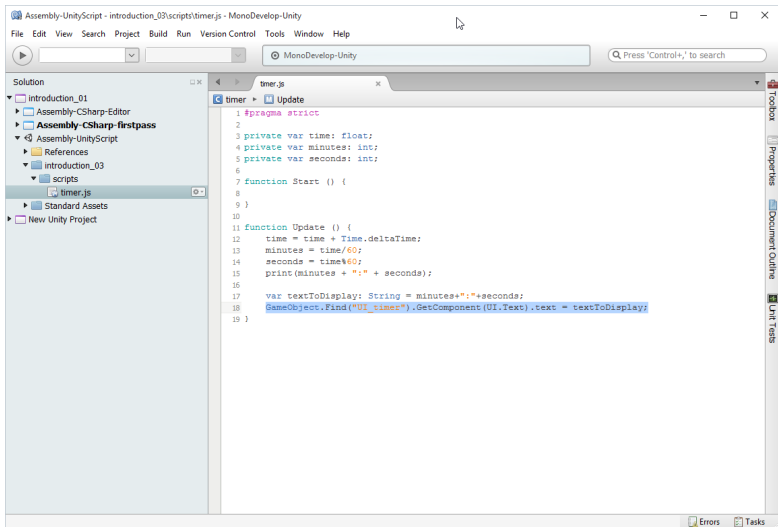


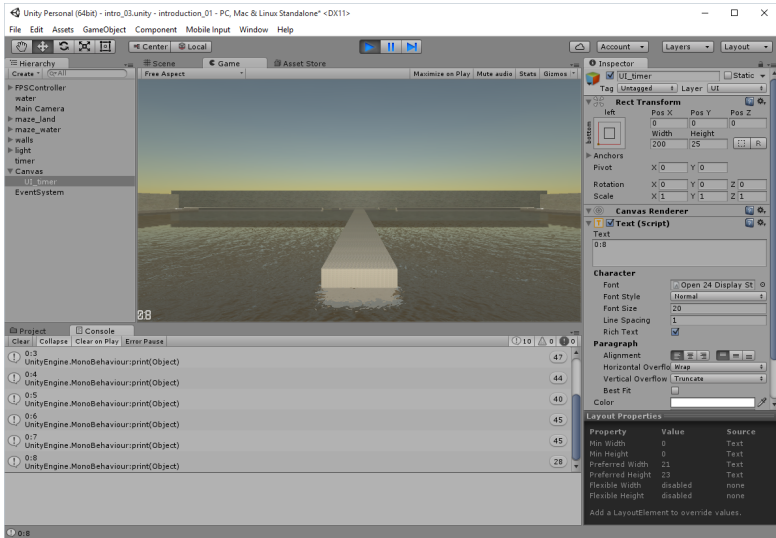
Displays the time on the screen

[Link timer script to the UI.Text](#)

We get an access to the object **UI_timer** and its **GUIText** component, and modify its attribute **text**. Add the following text at the end of the Update function

```
function Update() {  
    ... some existing code ...  
    ... add the following at the end of the Update ...  
  
    var textToDisplay:String = minutes+":"+seconds;  
    GameObject.Find("UI_timer")  
        .GetComponent(UI.Text)  
        .text = textToDisplay;  
}
```





Displays the time on the screen

Code tuning

- 1 Change the **timer.js** script code as follow

```
#pragma strict

private var time: float;
private var minutes: int;
private var seconds: int;

//private var uiTextToDisplayTime: GameObject;
private var uiTextToDisplayTime: UI.Text;
private var textTime: String;

function Start () {
//uiTextToDisplayTime = GameObject.Find("UI_timer");
uiTextToDisplayTime = GameObject.Find("UI_timer")
    .GetComponent(UI.Text);
}
```

- 2 To be continued – see next slide.

Displays the time on the screen

Code tuning – continuation

... continuation of the previous slide

- 1 Change the **timer.js** script code as follow (continuation)

```
function Update () {  
  time = time + Time.deltaTime;  
  minutes = time/60;  
  seconds = time%60;  
  textTime = minutes + ":" + seconds;  
  //uiTextToDisplayTime.GetComponent(UI.Text)  
  .text = textToDisplay;  
  uiTextToDisplayTime.text = textTime;  
}
```

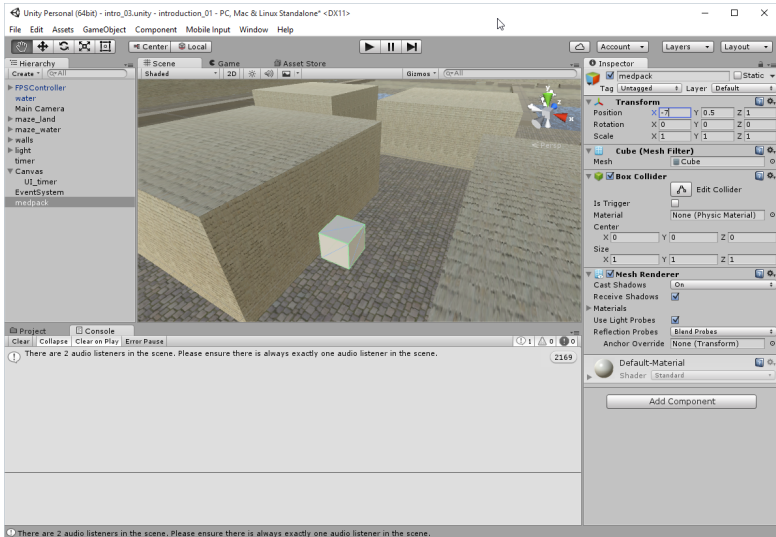
- 2 Play the scene and check that our code works properly.

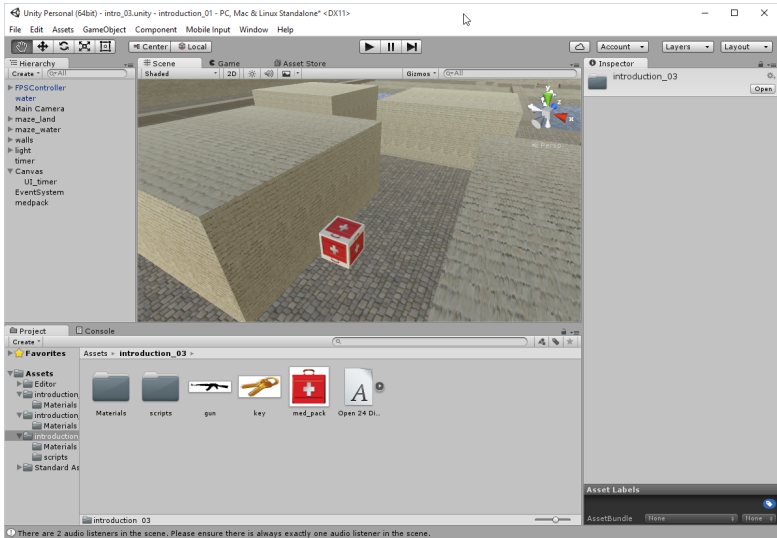
- 1 Create a new cube, rename it `medpack`, then change its position somewhere in the middle of the land maze (in my case: position $(x=-7, y=0.5, z=1)$, and its scale $(x=1, y=1, z=1)$).
- 2 Apply `medpack` texture to the cube.
- 3 Create a new script inside the folder `introduction_03 | Scripts` and rename it `rotate_medpack`.

- 4 Open this script and add the following code to Update function:

```
function Update ()
{
    transform.Rotate(Vector3(0,1,0), 90*Time.deltaTime);
}
```

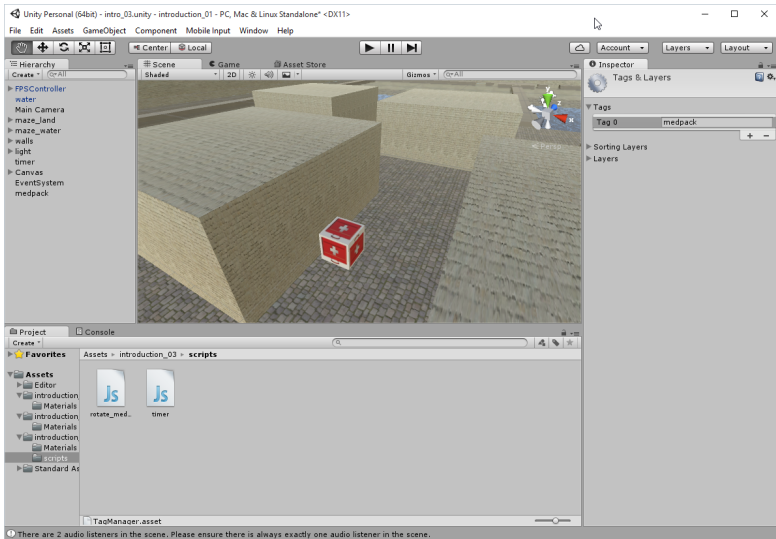
- 5 Link the script to the object labeled **medpack**, play the scene and check that the med pack is rotating.

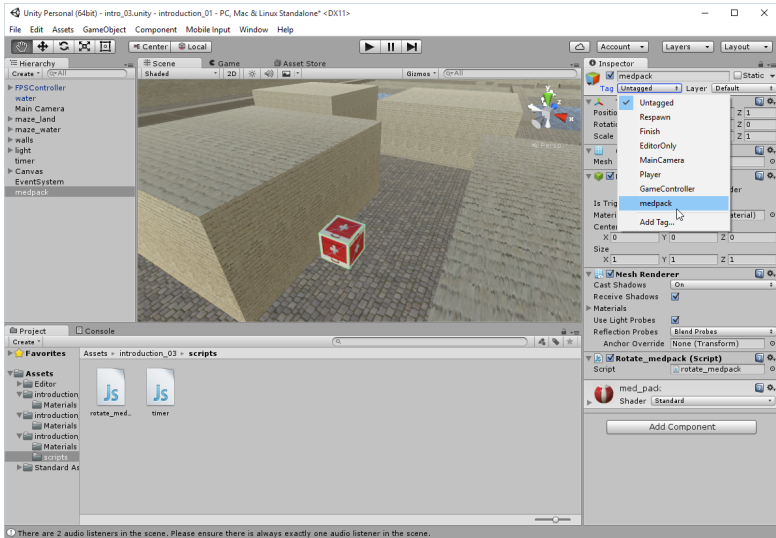




Add a tag so we will have an information on the collider involved in the collision.

- 1 Select the object labeled **medpack** in the **Hierarchy** window.
- 2 In the **Inspector** window, click on the drop-down menu to the right of the label **Tag**.
- 3 From the drop-down menu, select the option **Add tag**.
- 4 This should open a **Tags & Layers** tab. This will display a list of elements (or tags) available.
- 5 Click the plus at the bottom of the tags list, type medpack, and press **Enter**. This will create a new tag named medpack.
- 6 To apply this tag, click on the object labeled **medpack** in the **Hierarchy** window and click on the drop-down menu to the right of the label **Tag** in the **Inspector** window. This time the new tag **medpack** should appear.
- 7 Click on this tag to select it for the object.



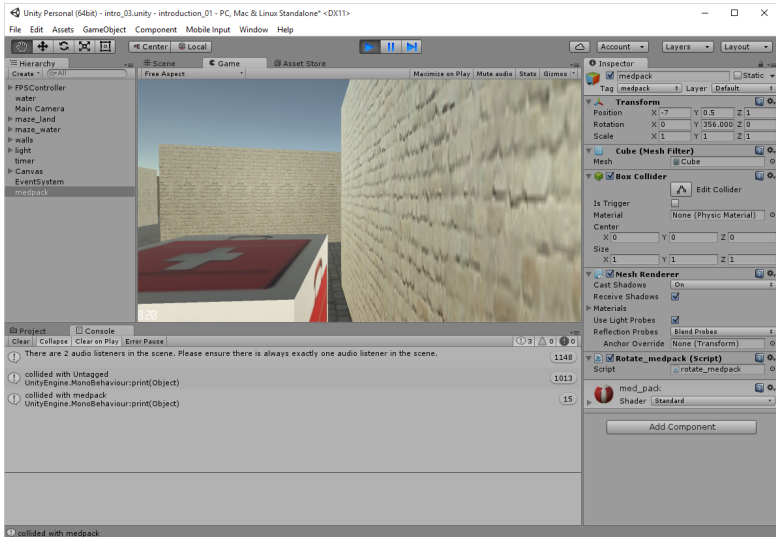


- 1 Create a new script inside the folder **Assets | introduction_03 | scripts** and rename it `collision_detection`.

- 2 Add the following code to the script:

```
function OnCollisionColliderHit(c : ColliderHit)
{
    print("collided with " + c.gameObject.tag);
}
```

- 3 Attach the script to the **First Person Controller** object, open the Console window (**Shift + Ctrl + C**) and test the scene (**Ctrl + P**). After colliding with the med pack, we should see a message in the **Console** window saying *collided with medpack*.



Collecting objects

Detect collisions – some remarks

Because the player is constantly walking (and colliding with the ground), and that the ground has no tag assigned yet, the **Console** window will display the message *collided with untagged*. Because this collision happens constantly (unless the player is jumping), the **Console** window may be flooded with messages. We may enable the option collapse in the **Console** window (button located at the top-left corner of the **Console** window); this will prevent messages from being displayed repeatedly and collapse identical messages accordingly.

Collecting objects

Destroy the med pack

We will now modify the script to destroy the med pack.

- 1 Add the following code to the script:

```
if (c.gameObject.tag == "medpack") Destroy(c.gameObject);
```

Add more colliding object

- Add **keys** object (box).
- Add **gun** object (box).
- Add...what you want.
- Attach the script labeled rotate to all of them.

Creating and displaying an inventory system

In our game, in addition to med packs, we will be able to collect other types of objects. We will then need to keep track of these objects using variables and graphical representations. This can be done using a basic inventory system. Some objects will have an effect on the player (for example, increase health), while other objects will be used at a later stage. To keep track of these objects we will need to create corresponding variables, update these variables when the corresponding objects have been collected, display a graphical representation of the object(s) collected, and modify the players' attributes (for example, its health). We will be working with the script `collision_detection`. First, let's create variables for the objects to be collected and add the following lines at the start of the script:

```
private var hasKey : boolean;
private var hasGun : boolean;
... variables for other object ...
private var health : int;
```

Creating and displaying an inventory system

Create tags for new colliding objects

- 1 Add tags for all object you have just added as we did it before for **medpack**.

Creating and displaying an inventory system

Different actions for different objects

```
function OnCollisionHit(c : ColliderHit)
{
if (c.gameObject.tag == "medpack"
|| c.gameObject.tag == "key"
|| c.gameObject.tag == "gun")
{
print("collided with " + c.gameObject.tag);
Destroy(c.gameObject);

if (c.gameObject.tag == "medpack"){
health = 100;
}
else if (c.gameObject.tag == "key"){
hasKey = true;
}
else if (c.gameObject.tag == "gun"){
hasGun = true;
}
}
}
```

Creating and displaying an inventory system

Code for displaying notification message for a few seconds

Now we will create a script named `display_message_to_user` that will display a notification message about new object on the screen and hide it after few seconds. Add the following code to the script:

```
private var timer: float;
private var displayTime: float;
private var timerIsActive: boolean;
private var message: String;
private var uiText: UI.Text;
```

```
function startTimer()
{
    timer = 0.0f;
    uiText.text = message;
    timerIsActive = true;
    displayTime = 3.0f;
}
```


Creating and displaying an inventory system

Code for displaying notification message for a few seconds – continuation

...continuation of the previous slide

```
function Start () {  
    // We can do this either as we did it before  
    //uiText = GameObject.Find("UI_displayMessageToUser")  
    //          .GetComponent(UI.Text);  
    // or like this  
uiText = GetComponent(UI.Text);  
}
```

Creating and displaying an inventory system

Code for displaying notification message for a few seconds – continuation

...continuation of the previous slide

```
function Update()
{
    if (timerIsActive)
    {
        timer += Time.deltaTime;
        if (timer > displayTime){
            timerIsActive = false;
            uiText.text = "";
        }
    }
}

function displayText(mes:String)
{
    message = mes;
    startTimer();
}
```

Creating and displaying an inventory system

Code for displaying notification message for a few seconds

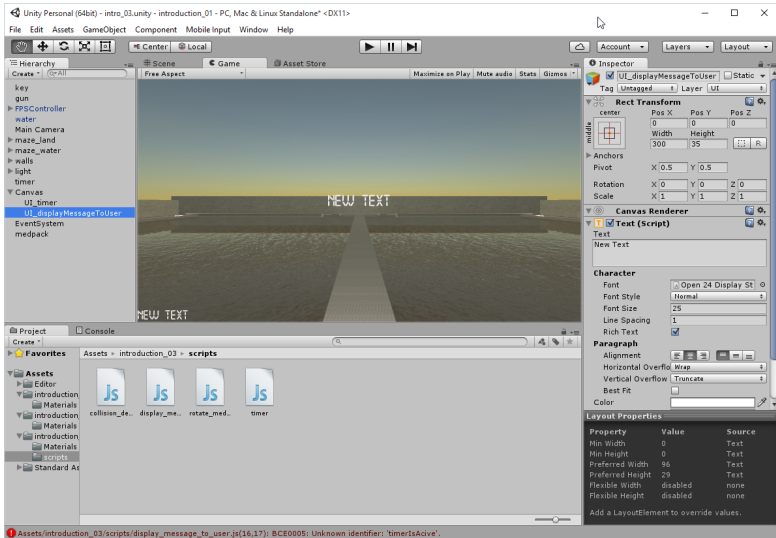
- 1 Create a **UI.Text** object.
- 2 Rename it `UI_displayMessageToUser`.
- 3 Change its position to be displayed in the centre of the screen. You can also change other attributes like font, color etc.
- 4 Attach the script `display_message_to_user` to this object.
- 5 Call the function `displayText` whenever the player collects an item – modify the script `collision_detection` and add the following code after the line that starts with `Destroy(c.gameObject)`:

```
GameObject.Find("UI_displayMessageToUser")  
  .GetComponent(display_message_to_user)  
  .displayText(c.gameObject.tag + " collected!");
```

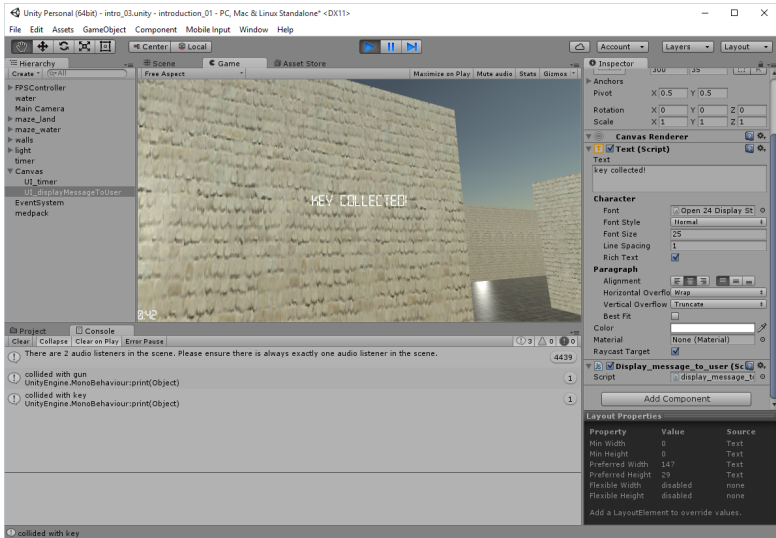
- 6 Adding the following line of code in the `Start` function of the script `display_message_to_user`.

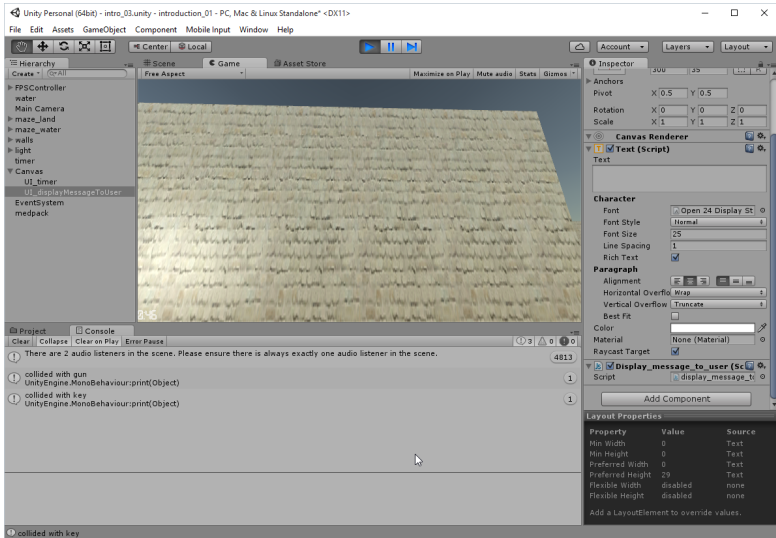
```
uiText.text = "";
```

to remove default text being displayed.



Assets/introduction_03/scripts/display_message_to_user.js(16,17): BCE0005: Unknown identifier: 'timerIsActive'.

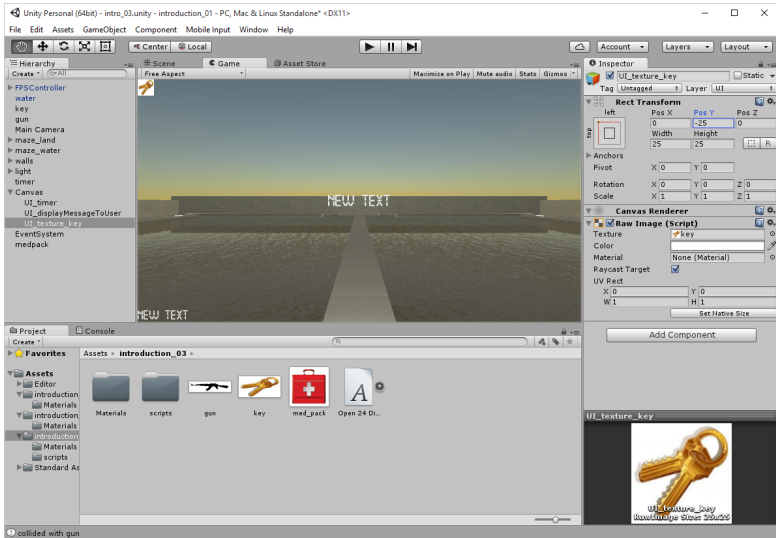


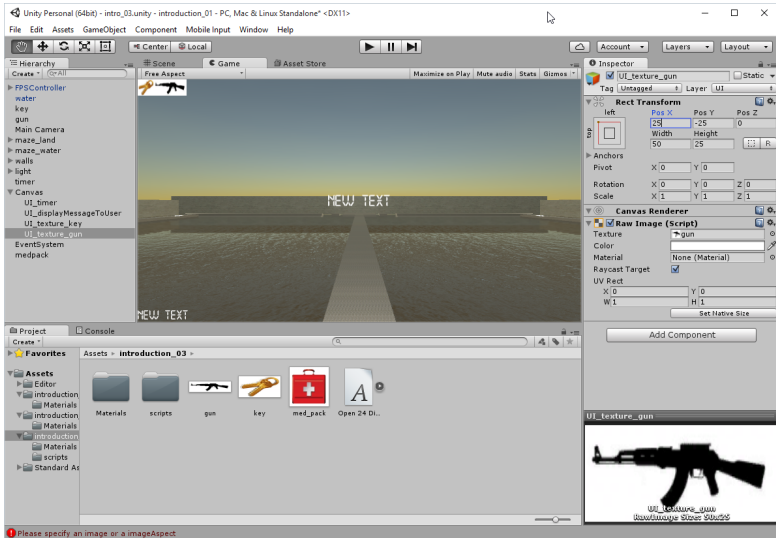


Creating and displaying an inventory system

Create texture for icons

- 1 Create a new **UI.RawImage** object (**GameObject** | **UI** | **Raw Image**) which we will use to hold texture.
- 2 Rename this **UI.RawImage** object **UI_texture_key**.
- 3 Locate the icon texture for key in the folder **Assets** | **introduction_03**.
- 4 Check that the **Inspector** window of **UI_texture_key** object is visible.
- 5 Drag-and-drop this texture in the **Inspector** window, to the right of the label **Texture** in the **Raw Image** component of the object **UI_texture_key**.
- 6 Using the **Inspector**, change the position of this object to display it in the upper-left corner of the screen.
- 7 Repeat the previous steps for all other objects.





Creating and displaying an inventory system

Display icons

Add the following code to collision_detection script:

```
function changeUITexture(what: String, display: boolean){
    GameObject.Find("UI_texture_" + what)
        .GetComponent(UI.RawImage)
        .enabled
        = display;
}
```

Creating and displaying an inventory system

Display icons

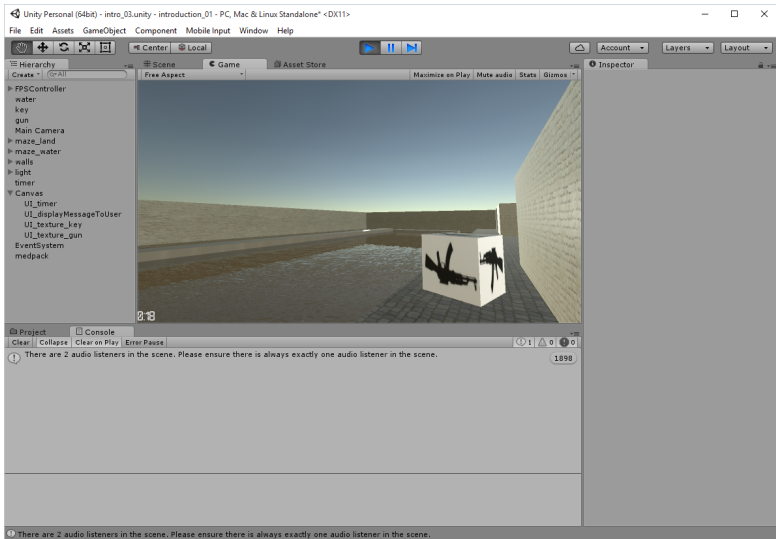
Modify the code according to the following snippet:

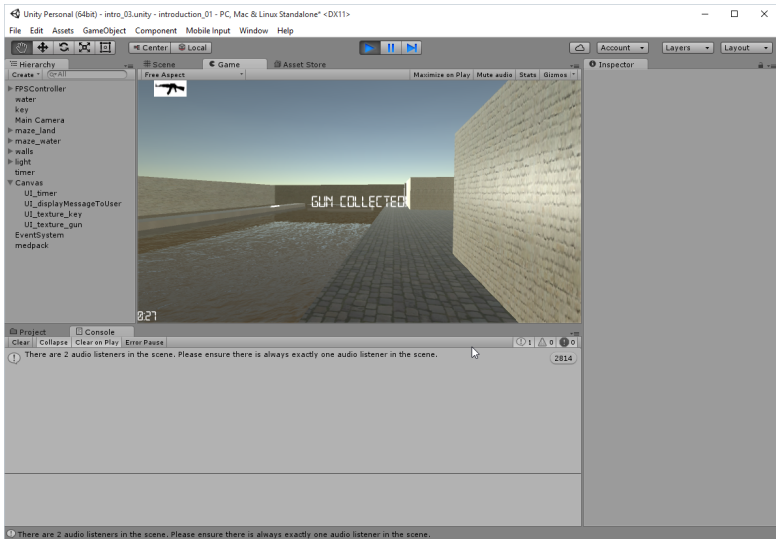
```
function OnControllerColliderHit(c : ControllerColliderHit){
    ...
    if (c.gameObject.tag == "medpack"){
        health = 100;
    }
    else if (c.gameObject.tag == "key"){
        hasKey = true;
        changeUITexture("key", hasKey);
    }
    else if (c.gameObject.tag == "gun"){
        hasGun = true;
        changeUITexture("gun", hasGun);
    }
    ...
}
```

Creating and displaying an inventory system

Hide icons at the start of the scene

```
function Start () {  
    hasGun = false;  
    hasKey = false;  
    health = 0;  
    changeUITexture("key", hasKey);  
    changeUITexture("gun", hasGun);  
}
```





There are 2 audio listeners in the scene. Please ensure there is always exactly one audio listener in the scene.

Finishing the game

- 1 Create a new tag titled `exit_door`.
- 2 Add the tag to the object named `exit_door`.
- 3 Modify the script `collision_detection` by adding the following lines within the function `OnControllerColliderHit`:

```
function OnControllerColliderHit(c : ControllerColliderHit){
  if (c.gameObject.tag == "medpack"){
    ...
  }
  else if (c.gameObject.tag == "exit_door"){
    if (hasKey){
      GameObject.Find("UI_displayMessageToUser")
        .GetComponent(display_message_to_user)
        .displayText("Great!!! You have just completed the game level");
    }
    else {
      GameObject.Find("UI_displayMessageToUser")
        .GetComponent(display_message_to_user)
        .displayText("Sorry, you need the key to open this door");
    }
  }
}
```

Unity Personal (64bit) - intro_03.unity - introduction_01 - PC, Mac & Linux Standalone* <DX11>

File Edit Assets GameObject Component Mobile Input Window Help

Center Global

Hierarchy Scene Game Asset Store Inspector

Creates - All Free Aspect Maximize on Play Mute audio Stats Gizmos

FPSController
water
gun
Main Camera
maze_land
maze_water
walls
light
timer
Canvas
UI_timer
UI_displayMessageToUser
UI_texture_key
UI_texture_gun
EventSystem
medpack

851

GREAT!!! YOU HAVE JUST COMPLETED THE GAME LEVEL!!!

Project Console

Clear Collapse Clear on Play Error Pause 5387

There are 2 audio listeners in the scene. Please ensure there is always exactly one audio listener in the scene.

Inspector

Tag Untagged Layer UI

Rect Transform

center
middle

Pos X 0 Pos Y 0 Pos Z 0
Width 300 Height 70

Pivot X 0.5 Y 0.5

Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer

Text (Script)

Text
Great!!! You have just completed the game level!!!

Character

Font Open 24 Display ST
Font Style Normal
Font Size 25
Line Spacing 1
Rich Text

Paragraph

Alignment
Horizontal Overflow Wrap
Vertical Overflow Truncate
Best Fit
Color

Layout Properties

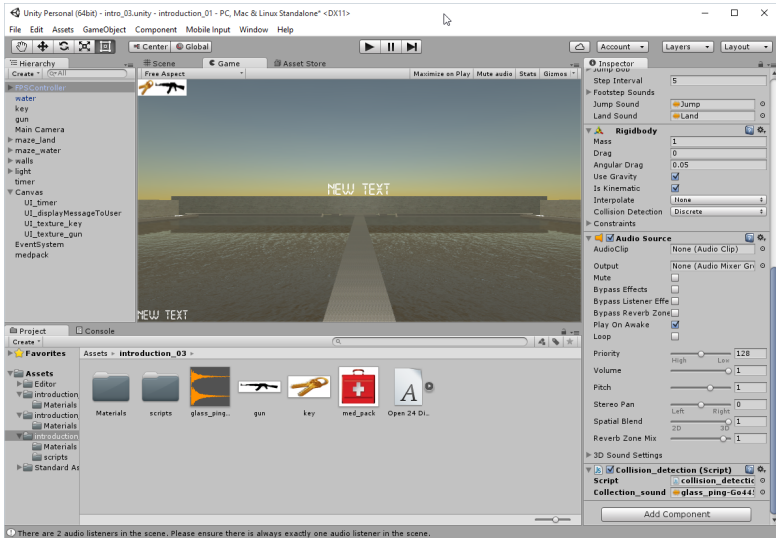
Property	Value	Source
Min Width	0	Text
Min Height	0	Text
Preferred Width	526	Text
Preferred Height	59	Text
Flexible Width	disabled	none
Flexible Height	disabled	none

Add a LayoutElement to override values.

There are 2 audio listeners in the scene. Please ensure there is always exactly one audio listener in the scene.

- 1 Open the script `collision_detection`.
- 2 Add / change the following lines at the start of the script

```
// use for old version
//@script RequireComponent (AudioSource)
#pragma strict
public var collecting_sound: AudioClip;
```
- 3 Import the sound file to the folder **Assets | introduction_03**.
- 4 In the **Hierarchy** window, click on the **FPSController** object. In the **Inspector** window, drag-and-drop the sound file to the variable `collecting_sound` in the component **Collision_detection** of the object **FPSController**.
- 5 Check that the **FPSController** object is still selected.
- 6 Check if **Audio Source** component is available. If not, select **Component | Audio | Audio Source**. This should add an **Audio Source** component to the **FPSController** in the **Inspector** window.
- 7 Check that the option **Play on Awake** is not selected for this component.



Adding audio

Play the sound on collision

- 1 Open the script `collision_detection`.
- 2 Add the following line to the script:

```
function OnControllerColliderHit(c : ControllerColliderHit){
  if (c.gameObject.tag == "medpack"
  || c.gameObject.tag == "key"
  || c.gameObject.tag == "gun")
  {
    Destroy(c.gameObject);
    var audio: AudioSource = GetComponent.<AudioSource>();
    audio.clip = collecting_sound;
    audio.Play();

    ...
  }
}
```

You should know

- how to create scripts using JavaScript,
- how to detect collision.