

Game Programming

## **Final year project: SCP Foundation Papers Please**

### **Name ideas: Security Guard Simulator**

**Tutorial 1:** Clicking an object to make another object appear (*Example: Clicking a document to inspect it*)

#### Prerequisites

Before approaching this tutorial, you will need a current version of Unity and a code editor (such as Microsoft Visual Studio Community) installed and ready to use.

This tutorial was created with Unity 2022.3 LTS and Microsoft Visual Studio Community 2022 versions. It should work with earlier or later versions. But you should check the release notes for other versions as the Editor controls or Scripting API functions may have changed.

If you need help installing Unity you can find many online tutorials such as:

<https://learn.unity.com/tutorial/install-the-unity-hub-and-editor>

You will also need to know how to create an empty project, add primitive objects to your scene, create blank scripts, and run projects from within the editor. If you need help with this, there is a short video demonstrating how to do all these things here:

<https://www.youtube.com/watch?v=eQpWPfP1T6g>

#### Objectives

In this tutorial, we will write a script that takes player input in the form of clicking an object and use it to make another object appear.

<https://youtu.be/JCCA9eaoVFc>

#### Getting started

To begin with, create a new Unity project, add a square object, and then create a new script called *bigSquare*. Make sure your new script is added to the square object.

It's a good idea to run the project at this stage to make sure there are no errors or problems to deal with before starting to code.

#### Starting the script for the big square

For this I have not removed the start script, but you can remove it if you want. We also don't need the lines of code at the top apart from "using UnityEngine;". Without this, our script will not work.

We start with "public class bigsquare : MonoBehaviour". Let's break this down.

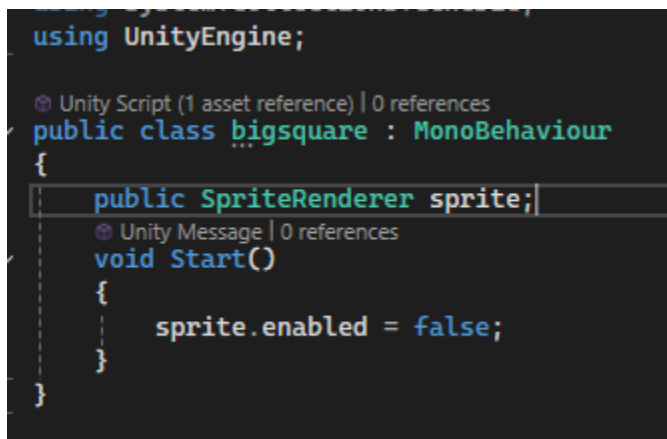
The "public" refers to the fact that we don't want this to be private and tethered to only one script.

The "class" is a container for variables.

The "bigsquare" is the name of the variable.

The ":" is used to define the type of a variable.

The "MonoBehaviour" is a base class for every script that provides a framework for attaching scripts to GameObjects and offers life cycle functions.

A screenshot of the Unity Inspector window showing the code for a script named 'bigsquare'. The code is as follows:

```
using UnityEngine;

public class bigsquare : MonoBehaviour
{
    public SpriteRenderer sprite;
    void Start()
    {
        sprite.enabled = false;
    }
}
```

The 'bigsquare' class is highlighted in blue. The 'SpriteRenderer' variable is highlighted in green. The 'Start()' method is highlighted in blue. The 'sprite.enabled = false;' line is highlighted in blue.

Inside the brackets is "public SpriteRenderer sprite;". We know what "public" means but SpriteRenderer means it will render the assigned Sprite.

Then, "sprite.enabled = false;". This is an example of a Boolean script, because of the "false". "enabled" refers to the visibility of the sprite. We want it to be invisible at the start. That is all we need in this script, as everything else will be outlined in our other script.

Let's check if this works. If you run the game and the big square disappears, you have been following along well.

### Starting the script for the small square

Now add a new sprite for the small square. Make sure it doesn't overlap with our other sprite.

Again, we don't need the Start (and Update) sections of the script.

```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class smallsquare : MonoBehaviour
{
    public SpriteRenderer bigSquare;
}
```

Remember to include “using UnityEngine;”. And then we do as we have done before with the next lines.

Further down there will be written “void OnMouseDown()”. You can likely guess what this means just from reading it, it is describing what happens when the input of the sprite being clicked by a mouse is received. But to break it down:

You use void as the return type of a method (or a local function) to specify that the method doesn't return a value.

Then, OnMouseDown is just how you write the input of a mouse click.

In continuation, you must add these brackets “{}” underneath this line of code. Within the brackets is where we will type our next line. This will be “bigSquare.enabled = true;”.

Our scripts should be complete, try running the project and clicking on the small square. If the big square appears you have been successful.

This doesn't need to just use squares. You can replace these sprites with anything that you want. For example, if you are making a point and click game, say it is an asset of a document. Clicking the document makes a screen appear where you can examine what is written on it. That is one way you can utilize this tutorial.



## Tutorial 2: Functioning menu

### Objectives

In this tutorial, create a functioning menu.

<https://www.youtube.com/watch?v=CxXIJKLde2Y>

### Getting started

To begin with, create a new Unity project, and add two buttons. Line them up vertically.

It's a good idea to run the project at this stage to make sure there are no errors or problems to deal with before starting to code.



Now we create a script, call it “Menu”. For this script we will need to add “using UnityEditor;” so that it works.

Then, we type “public class Menu : MonoBehaviour”. Menu is our variable. We open brackets and say, “public void Selected()”. This will target the “Continue” button.

Another square bracket is needed and then “gameObject.SetActive(false);”. A gameObject is the base class for all entities in unity in the scene hierarchy. “SetActive” refers to the visibility of the buttons. And we want it to be false when the continue button is pressed.

Below, “public void Quit()”. This is what happens when the “Quit” button is pressed. Brackets, and then “#if UNITY\_EDITOR”. We will be writing an if statement. This line means, if the game is being run in the Unity Editor, then...

Then, “EditorApplication.ExitPlaymode();”. EditorApplication is the Editor, and ExitPlaymode is how you stop the game from running. Next is “#else”, “Application.Quit()”. So, if the game is being run after it has been built, you have to close it in a different way, this way. “Application” is the game. And “Quit” is how you close it.

Don’t forget to end the if statement with “#endif”.

In the Unity Editor click on “button1” and go to the “Button” tab. We go to “On Click ()” and make sure the script is put in like this:



<https://www.youtube.com/watch?v=lB9TGzgtQFg>

### Getting started

To begin with, create a new Unity project, and add a square to the project.

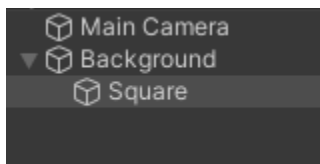
It's a good idea to run the project at this stage to make sure there are no errors or problems to deal with before starting to code.



Now, create a script and call it “Square”. We start with “public GameObject square;” so we have a reference to the square sprite. Then, “void OnMouseDown()” to detect mouse clicks.

Inside the brackets we put “square.SetActive(false);” to make the square disappear.

Make another sprite and call it “Background”, it will be a square except you transform it to fit the entire screen. Change the opacity of this sprite to 1. We want it barely visible, unfortunately this won’t work if the value is at 0. Make sure your hierarchy looks like this:



Attach the “Square” script to the background. Then, add “Box Collider 2D” to both the square and the background so it can recognise being clicked.

There is an issue with this. Because both the square and the background have the same Z value, it will not recognise a click for the square. You must set the square’s Z value to -1.

Now try it. The square should disappear when you click outside of it and stay visible when you click on it.

This can be used to stop viewing a document in a game.



#### **Tutorial 4:** Translating a character into the scene

##### Objectives

In this tutorial, we will create a character sprite, and make it move across the screen.

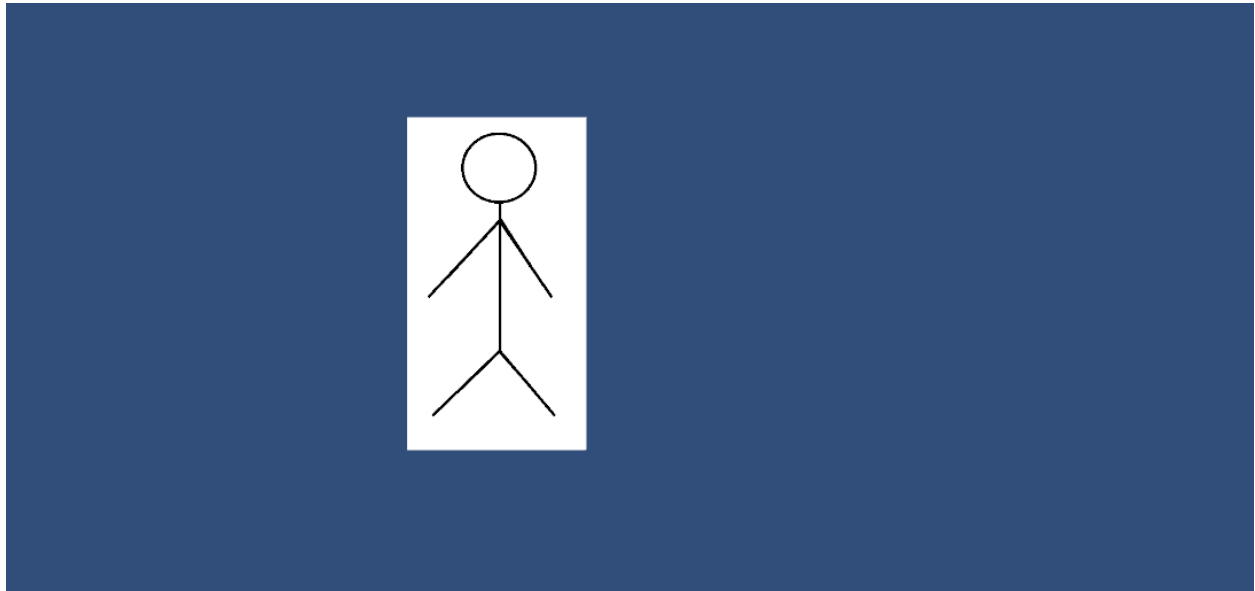
<https://youtu.be/s2MAdR65kGA>

##### Getting started

To begin with, create a new Unity project, and add a sprite that can be used for the character. For this tutorial I have used a basic stick figure.



It's a good idea to run the project at this stage to make sure there are no errors or problems to deal with before starting to code.



We can move it off-screen so that we can code it to move on-screen. We may need to script its starting position.

Create a new script “Character”. Underneath “void Start()” type “transform.position = new Vector3(-13, 0, 0);”. This outlines the starting position of the character; transform is used when we do things to a sprite, for example moving it to a specific coordinate. “position” means we are changing the position of the sprite. “Vector3” is just how we put coordinates, with the brackets being the x, y, and z coordinates.

We can test this out. Move the character anywhere within the play area in the Editor, then press play. If it’s not visible it means, it’s offscreen which is what we want.

Now we have to make it move across the screen.

We will be using a coroutine, we could use an if statement, but this way will be easier. Underneath the line of code that sets the sprite’s starting position we write “StartCoroutine(Animate());”. StartCoroutine is how we start a coroutine and “Animate” is what we are going to do to the sprite.

Then, outside of “void Start()” we can put “IEnumerator Animate()”. The “IEnumerator” is needed for a coroutine. Then open brackets, “while (transform.position.x < 0)”. This is a while loop, which is basically saying “while the x value is lower than 0”, zero being the middle of the screen.

Open brackets, “transform.position += Vector3.right \* Time.deltaTime \* speed;”. This is saying we are changing the position of the sprite so that it moves to the right at a certain speed. Now we shouldn’t forget to end this with “yield return null;”, otherwise this won’t work. That means we return control back to Unity.

```
transform.position = new Vector3(-13, 0, 0);
StartCoroutine(Animate());
}

// Update is called once per frame
1 reference
IEnumerator Animate()
{
    while (transform.position.x < 0)
    {
        transform.position += Vector3.right * Time.deltaTime * speed;
        yield return null;
    }
}
```

Try running the project, does the sprite slide onto the screen and stop at a certain point? If so, you have completed this tutorial successfully.

---

References:

<https://github.com/LSBUGPG/movement-tutorial>