**Part 1: Make A Face**

The first step is to create a new project. Open up Unity and select *new*. After that you’ll be taken to a screen where you can name the project and select it’s type. Select 2D and when everything looks good, click on *Create Project*

**INSERT PIC HERE**

Now you’ll be in the project window. It may look scary at first, but don’t worry, I’ll go easy on you. To start we need to create the folders to hold our files. We can start with three easy ones, *Sprites*, *Script*, and *Prefabs*. To create these, right click the area within the red rectangle in the picture below and select Create > Folder and then input the name. After doing this three times, you should see the three folders at the bottom of the screen.

**INSERT PIC HERE**

Now we need to fill these folders. Double click inside the *Scripts* folder to enter it, and then create a new C# script inside of it. To do this, perform the same process you used to create the folders, except click *C# Script* instead of *Folder*. Name this script *MouseManager*. This script will be used to manage data related to our mouse pointer. If you have ever used Java before, it may be easier to understand if you imagine this script as a *Mouse* class.

Before we edit the script, we are going to create an object in Unity that represents our mouse. To do this, right click inside the *Hierarchy* window on the left side of the screen and select Create Empty. The new object should appear with the name *GameObject*. That name is not very descriptive (or fun) so right click it and rename it to something fun like *Jerry*.

**INSERT PIC HERE**

Next we have to attach our script to this object so the script can actually do something instead of just looking pretty. To do this, drag and drop the *MouseManager* script at the bottom of the screen onto the *Jerry* object on the left side of the screen. If done correct, the *Inspector* window on the right side of the screen should look something like the picture below. Our object will contain two different components, *Transform* and *MouseManager*. We have already established what *MouseManager* is, but what is *Transform*? *Transform* is a component that comes default with all Unity objects, and it is what controls where on the 3D plane our object is. We will be manipulating it very soon, but for now double click the script at the bottom of the screen to open it in a text editor.

**INSERT PIC HERE**

You can use whatever text editor you want. I am using Microsoft *Visual Studio*, but I believe the default editor for Unity is MonoDevelop. You’re screen may look a little different than mine, but that’s ok, it’s about what’s on the inside, not what’s on the outside.

**INSERT PIC HERE**

You’ll see the class name we entered earlier, *MouseManager*, at the top of the screen followed by a colon and the name *Monobehaviour*. This means our class inherits *Monobehavior*, which allows our script to be attached to objects in Unity. You’ll see how this works later. Our class comes default with two incredibly useful methods, *Start* and *Update*.

*Start* is used to initialize the object’s fields, similar to class constructors in languages like Java and C#. It’s not the exact same thing, but it’s a good way to simplify it. This method will be automatically called when the scene initially starts playing in Unity as long as it is attached to an active object.

*Update* is used to…. Update. Every frame, this method is called in every single script that implements it. This means that if we have an object in Unity that attaches a script that has *Update* method, the object will be constantly updating every frame (and there are A LOT of frames).

First order of business is to get the position of the mouse so we can make our in game mouse object follow it. Declare a new *Vector3* object above *Start* and name it *mousePosition*. This will hold the value of our computer cursor. Then in *Start* method, initialize this by setting it equal to **new Vector3(0, 0, 0)**. Vector3 is a Unity object that is used to represent positions on a 3D plane. It has 3 coordinates, X, Y, and Z. Since we are making a 2D game, we will only be focusing on X and Y at the moment.

Now that we have initialized our *mousePosition* value, we can try setting it. We will start by creating a new method, *SetPosition*. Make this method have a return type of void since it will not need to return any values and make it private so it can only be accessed by our main man *Jerry*. Your method should look something like **private void SetPosition()**. First thing this method will do is get the position of our cursor. We will be using some included Unity scripts to do this. In the method, set *mousePosition* equal to **Camera.main.ScreenToWorldPoint(Input.mousePosition)**. Now every single frame, *mousePosition* will be set to the position of our cursor.

Now that we have the position of the mouse, we have to set *Jerry*’s position equal to that so they can always be together. To do this, we have to set the position of *Jerry*’s *Transform* component. To do that, enter **transform.position = mousePosition**. *transform.position* is the *Transform* component’s position value and *mousePosition*  is the mouse cursor position. Next, the line **Debug.Log("Mouse Position: " + transform.position.ToString());** to the bottom of the method. This will print Jerry’s position so we can make sure everything is working. Lastly, add a call to *SetPosition* inside the *Update* function so that it will be called every frame.

**INSERT PIC HERE**

Now save that script by selecting File > Script in the tool bar and then head back into Unity. Click the play button at the top of the screen (pictured below) to run the scene so we can test our script. As you move your cursor on the screen, you should see the values in the console at the bottom of the screen change to reflect the position of the cursor. Neat!

**INSERT PIC HERE**

**INSERT PIC HERE**

This is cool and all, but it doesn’t really do much on it’s own. Let’s makes things a little more fun and give our old pal *Jerry* a face. To do this, you will use the imported .PNG file below and Unity’s built in Sprite Renderer component. To start, download the file below and then drag and drop it into the *Sprites* folder you created earlier. You should see the picture appear in your Unity project files on the bottom of the screen when you select the *Sprites* folder. You have now successful imported your first **Asset** into Unity, congrats!

**INSERT PIC HERE**

Let’s add this new image to our Unity object. Select *Jerry* in the *Hierarchy* panel on the left. After it is selected, you’ll see its two components from earlier appear on the right side of the screen. Select *Add Component* and then type in and select Sprite Renderer. Now your object will have a blank sprite renderer. You can see a variety of options in the Sprite Renderer. The fun ones we’ll be focusing on are *Sprite*, *Color*, and *Flip*. Drag and drop the .PNG file from the *Sprites* folder onto the *Sprite* section of the Sprite Renderer. You should now see the .PNG file appear in the Unity scene. If you don’t, try zooming in, he may be hidden behind the camera icon.

**INSERT PIC HERE**

Sure he’s there, but he looks a little tiny and blurry right? Let’s fix that. To do this, we will have to alter the import settings on the .PNG image. Select the image inside of the *Sprites* Folder. After selecting it, you should see a variety of options appear on the right side of the screen under the *Inspection* tab. Let’s change a few of those. To start, change *Pixels Per Unit* from 100 to 32. The imported .PNG is 32x32, so this should make it approximately one Unity unit (very nice alliteration) large. Next change *Filter Mode* from Bilinear to Point (no filter). This will make our image appear in Unity exactly as we designed it outside of Unity. Then, click *Apply* at the bottom of the settings. You should now see a larger blockier smiley face.

**INSERT PIC HERE**

Alright, now let’s hit play and see how things are looking… hmm…. Things aren’t looking too good. You really won’t see anything even though you can see *Jerry*’s coordinates changing. What gives? Well it’s pretty simple. While we are handling the cursors position in 2D, Unity actually handles movement in 3D. *So* while Jerry may be in the correct X and Y positions, his Z position is not correct. Since we find the position using the camera, our Z position will actually be set to the Z position of the camera. This is no good, because the camera doesn’t render things if they are too close, and being in the exact same position is DEFINITELY what I would call too close.

To fix this we need to go back and edit our code a little. We are still going to find the position of our mouse in every *Update*, but instead of just setting the position of *Jerry* to be that position, we are going to create a new more correct instance of Vector3 using the mouse position. We can use Unity’s imported Vector3 constructor to do this. It takes three variables, an X, a Y, and a Z. Just pass in the mousePosition’s X (mousePosition.x), the mousePosition’s Y (mousePosition.y), and a new Z (try 0) and then set transform.position equal to that. Don’t forget to save the script!

**INSERT PIC HERE**

Now let’s head back into Unity and see if that’s any better. Click Play and you can see that our little man will now follow our mouse cursor! Our very first game!!! (just kidding… unless you’re bored I guess… then this really is your very first game…)

**INSERT PIC HERE**

For bonus fun, try selecting *Jerry* and editing the *Sprite Renderer* component by changing the *Color* value. We can use this value to change the color of the white pixels in our sprite. Not enough fun? Try selecting the boxes next to *Flip* to flip the sprite. The *X* won’t do anything visible since the sprite is already mirrored along the X-axis, but selecting *Y* should provide some interesting results…

**INSERT PIC HERE**

**Part 2: Make A Fist**

A face is fun, but what’s more fun is a face that can hit things. If *Jerry* want’s to interact with things in the environment, he’s gonna have to hit