## Creating a Project and Opening Unity

* Open Unity Hub.
* Select “New Project”.
* From the list of templates, choose “3D”.
* Give your project a name on the right.
* Click “Create project”.

Basic Panels and Setting up a Scene

* Basic Unity panels.
  + How to arrange them from default.
  + I always move the Project panel to a column beside the Inspector and change it to one-column mode.
* Import the provided .unitypackage file.
  + Inspect the files it added using the Project panel and the Inspector Panel.
  + Look at the squirrel & animation.
  + Note that there are several files for each object. We will get to the differences soon!
* Drag and drop the squirrel **prefab** into scene.
* Create some ground for it.
  + Make a plane.
    - Rename it “Grass”.
  + Brief discussion of 3D co-ordinates and the **transform** part of every object.
  + Y is up. Setting to 0 position puts at ground level.
  + X,Z are for making it wide. Setting scale to large number makes it big!
  + Create a folder and new material for the grass.
    - In the Project panel, right click on “Assets”.
    - Choose “Create—>Folder”.
    - Name it “Materials”.
    - In Unity, we usually organise our files by *type*.
      * In the assets we imported we saw them in Materials, Meshes, Prefabs, etc.
      * We will do the same, but under “Assets”.
    - Right click “Materials”, and choose “Create—>Material”.
    - We can change the colour beside “Albedo” to change the main colour.
    - Drag this material onto the grass in the Scene panel.
    - If you have trouble, you can also drag it onto the “Grass” object in the Hierarchy panel.
    - Feel free to play around with the colour or other properties on the material.
    - If you select the Ground object in the Hierachy, you can now see your material at the bottom as well.
* Play around with the light.
  + Rotation tool.
* Position the camera:
  + Move your camera in the scene view by holding the right mouse button.
    - Move the mouse to rotate.
    - Press WASD to move around.
  + Find a nice view of your squirrel and a little space in front of it.
  + Note that in the Game panel the view is different! This is what we see if we run our game, so we want it to look like the Scene view.
  + Select the camera object, then from the menu choose Game Object—>Align with View.
  + Now look at the Game panel and see it’s the same!

## Let’s Code Something!

* Create a Scripts folder, just like we did for Materials.
* Right click and choose “Create—>C# Script”, and name it “Squirrel”.
  + C# is the main programming language for Unity.
  + If you have programmed before in another language, you will probably find the concepts very familiar.
  + If not, don’t worry! We will go slow and help you :)
* Just like we did with the grass material, let’s drag and drop it onto the squirrel.
* Select your squirrel and make sure the script is there.
* But, the script doesn’t do anything to start with. We actually have to write some code in it!
* Double-click the script file in the Project panel.
  + This opens a different program where you can edit the file!
  + Depending on your system, it might ask which program or open a different program than someone else.
  + By default, you should have Visual Studio Community installed with Unity.
  + But any similar program is okay!
  + Let us know if it will not open anything, and we will help.
* Look at our script to start with:
  + We don’t need to know all the details now, but let’s do a quick overview:
    - At the top, it says “using” some things. These are the names of some other code provided by Unity or someone else that you can use within your code.
    - public class Squirrel : MonoBehaviour —> This basically says our script is named Squirrel and it is a “mono behaviour”, which is a standard type of Unity script.
    - The section inside the first { and the last } is all the code belonging to the Squirrel.
    - There are two sections inside this: Start and Update.
* The Start() section is where we can do things at the very beginning of the game, when we hit the Play button. We will come back to that.
* The Update() section is where we can do things during the game, and it runs every single *frame*. The concept of “frames” —> like frames of a movie. This happens many times every second, and if we change things a little bit each frame it comes alive.
* Let’s try moving the squirrel forward:
  + transform.Translate(Vector3.forward \* Time.deltaTime);
  + “Transform” is a special name we can use to refer to the “Transform” information on our object.
    - Reminder: Look at the inspector. We have position, rotation and scale.
  + “Translate” simply means to change the position.
  + “Vector3” is the type of data we want. It’s three decimal numbers (or “floats”): x, y and z.
    - Reminder: y is up, x is side-to-side, and z is forward-back.
    - Vector3.forward is a nice short way of saying (0, 0, 1).
  + “Time.deltaTime” is important!
    - Unlike with a movie, the speed of the frames in our game may not always be the same.
    - A fast computer might have 60 frames per second. A slow computer might have 15.
    - If we move the same amount regardless, then on a fast computer our squirrel will run fast and on a slow computer our squirrel will run slowly. We don’t want this!
    - Fortunately, Unity tells us the exact amount of time between each frame. That is Time.deltaTime.
  + Notice that there is a mix of capitalisation here.
    - We will go into *why* soon.
    - But for now, remember that it is very important. If you type it wrong, it will not work!
  + Most software will underline typing mistakes in red.
* Back to Unity.
* Hit Play.
* It moves! Hurray!

## Interaction — Getting Input from the Player

* Control by holding key.
  + if (Input.GetKey(KeyCode.**UpArrow**))
  + Include {}, putting our previous line inside the new section.
  + The if(...) part is called a “conditional statement”. It means that only if something is true will the section after run. So we can control what code executes when.
  + Input.GetKey(...) —> This tells us if a key is down.
  + KeyCode.UpArrow —> This just means the up arrow on our keyboard.
* Run again inside Unity. Now the squirrel goes forward, only when we are pressing the up key.

## Some Technical Details (Theory)

* We typed lots of things, some with capitals, and some without. Some with () and some without. What does it all mean?
* Let’s break down our code a little. It’s okay if you don’t absorb all of this, as this is only an introduction, but if you can understand the different types of code here it will be easier to guess at how to do new things.
* public class Squirrel
  + A “class” is a **kind** or category of thing. Classes get capital names.
* void Update () and void Start()
  + These are **functions** we create in this script.
  + Functions do things. They are like verbs.
  + These two are special and Unity knows how to use them based on the name, even though we get to write what they actually do.
  + We can create many other functions if we want, but probably not in this workshop.
  + Functions get capital names, too.
* transform is a *variable*. It is a piece of information. Variables in C# are usually not capitalised on the first letter.
* Vector3 is a struct. Structs are very similar to classes, and also get capital names.
* Vector3.forward —> This is a variable belonging to the type Vector3. When a variable or function belongs to a class or struct we use the dot like this. Vector3.forward has the value (0, 0, 1).
* Time.deltaTime —> This is a variable on the Time class. It has a value, usually very small, measured in seconds. Something like 0.03333, but it can actually be different each frame.
* Input.GetKey(...) —> Again, a function.
  + But instead of *doing* something, this *returns* some value.
  + This is a very simple type of value called a bool, which is true or false. We will see bools again, soon.
* if (....) —> This is a key word in C#. Key words are always lower case.
  + Same: class, void, public, using
* KeyCode.UpArrow —> This is an enum.
  + Enums are like a group of possible values in a category. In this case, the possible keys that can be pressed.
  + “KeyCode” is the name of the enum.
  + “UpArrow” is one of the possible values.
  + Just like variables belonging to a class, we use the dot.

## Rotation

* Adjust speed by multiplying!
* Add rotation.
  + if (Input.GetKey(KeyCode.**LeftArrow**))
  + {
  + transform.Rotate(0, -90 \* Time.deltaTime, 0);
  + }
  + if (Input.GetKey(KeyCode.**RightArrow**))
  + {
  + transform.Rotate(0, 90 \* Time.deltaTime, 0);
  + }

## Animation Time!

* Create an animator controller.
* Add the animations we have. Stand, walk.
* We need to CONFIGURE walk to loop!
* Set the stand one to be default (gold).
* Draw arrows between.
* Create parameter “run”.
* Make conditions that use the parameter.
* Finally, we must add to our code to set the parameter!
  + private Animator myAnimator;
  + myAnimator = GetComponent<Animator>();
  + myAnimator.SetBool("Run", true);

## Time to Save!

(Because this is where Unity crashed while I for me :))

* Be sure to use both Ctrl-S to save your scene, and to choose File—>Save Project.

## Let’s Decorate! — Adding More Objects to Make the Scene Interesting

* Our first “level”.
* Just drop in a few trees or rocks.
  + We are making a “level” or “scene” using “prefabs”.
* Create a “mango” by starting with a sphere.
  + We can also create new objects from scratch in Unity, but the shapes are limited.

## Object Parenting — A Simple Camera Method

* Re-position the camera behind the squirrel. Same procedure as before.
* Parent to squirrel. This makes it stay with the squirrel no matter where it goes!

## Tweaking Speed — Values We Can Edit in the Inspector

* Our first “serialised” property. This let’s use configure in the inspector!
* [SerializeField] private float **speed** = 5;
* Float means “floating point number” and is a fancy computer programming way of saying a number that can have a decimal.
* Demonstrate that we can run the game and change this value in the inspector, even while the game runs.
* Warning: When you stop the game, the value resets.

## Picking up the Mango — Using Colliders

* Sphere collider on mango as “trigger”.
* Sphere collider AND rigidbody set to kinetic on squirrel.
* Add a Mango script:
* private void **OnTriggerEnter**(Collider other)
* {
* GameObject.Destroy(gameObject);
* }
  + We can ignore the “Collider” part here. It would let us do even fancier things.
  + “GameObject.Destroy()” is a Unity command to destroy some object.
  + “gameObject” is a special name for the object this script is attached to.
* Now when we touch the mango it disappears!

## Keeping Score — Public Variables & UI

* Let the squirrel keep score:
  + public int **Score** = 0;
  + “Public” means we can get it elsewhere.
  + Int means integer. No decimal. Our squirrel always eats the whole mango!
  + private void **OnTriggerEnter**(Collider other)
  + {
  + Score++;
  + }
* Add a UI!
  + Canvas.
  + View from “Back”. Play with scene view controls.
  + Create Text.
  + Unity will ask to import text mesh pro stuff. Annoying but okay.
  + Now we have a text element. By default, hard to read. Make it black, bigger, bolder.
  + Set the position using the special rect transform stuff.
  + Now a script:
    - First, we get another component on this object, which we did before with the squirrel’s animator. But this time we need to add a “using” statement because TMP is not included by default.
    - using TMPro;
    - private TMP\_Text scoreText;
    - scoreText = GetComponent<TMP\_Text>();
    - Then we need to actually know about the squirrel. We can use a serialised field here!
    - [SerializeField] private SquirrelMovement **squirrel**;
    - scoreText.text = $"Score: {squirrel.Score}";
* Duplicate your mango and see this working!

## Resources:

Some free animals:

<https://itch.io/game-assets/free/tag-3d/tag-animals>