Workshop 4: Digital Media
By Daniel Crippa

Topics Covered
- Introduction to the Unity Engine
- Components (Rigidbodies, Colliders, etc.)
- Prefabs
- UI
- Tilemaps
- Game Design

Introduction to Unity

Unity3D is a free game engine for the production of 2D and 3D games. It is available for both Windows PC and Macintosh and can publish games for a wide variety of platforms, including desktop, mobile and web and consoles.

Important: Unity frequently releases new Major and Minor versions which can greatly impact the way that the engine works, along with adding/removing features. Using different versions of Unity to make a project is not recommended as this likely results in your game crashing. In this workshop we will be using Unity 2017.3.1

Before we can begin in making a game using the Unity Engine we first need to have a basic understanding about its essential elements and functions. This section aims to introduce you to the essential functions of Unity.

When you open Unity for the first time you will be prompted to make an account and activate your license. Lucky for us a license has already been activated on these computers so just press the Work Offline or Skip button.
You will now be able to create a new project or open an existing one. For this exercise we will open an existing project as it already includes the basic assets we will be using later.

Press the **Open** button and locate the project folder **X** located at **X**. This may take some time to open as it will need to ensure all of the files are in a working condition. Once that has loaded you will be presented with the main window of Unity, this is where we will be doing most of our work.

Let’s have a brief look at what we’re given

1. **Toolbar:** These are the tools that you will be using to manipulate your objects in the scene view, these tools are all quite intuitive. From Right to left they are:

   a. **The Hand tool**, which allows you to move around the scene
   b. **The Translate tool**, which allows you to move objects
   c. **The Rotate tool**, which allows you to rotate objects
   d. **The Scale tool**, which allows you to scale objects
2. **Hierarchy**: Every object/asset that exists in your scene will appear in your hierarchy. This is a good area to select the specific object that you want to inspect.

3. **Scene/Game View**: The Scene view is where the placement and manipulation of objects in your game takes place, it is here where you will make use of the tools mentioned in the toolbar. The Game View is what your game actually looks like to players, when you test play your game this view will automatically be opened.

4. **Test Play Controls**: These tools allow you to test play your game, from left to right they are:
   - **Play**: Plays your game
   - **Pause**: Pauses the game
   - **Next Frame**: Moves to the next frame in your game

5. **Inspector**: This is where most of the work takes place. The inspector will allow you to manipulate specific components of an object in your scene. Components are scripts that are pre-programmed in the Unity Engine and of course, any that you write yourself.

6. **Project Panel/Console Window**: The project panel houses all of your assets that you will be using in your game, this includes: scene files, music, art, animations, etc. The console window assists in the debugging of your game, it’s a very useful tool but not one that we will be using today.

Now that you know the basic layout of the Unity Engine, let’s have a look at the specific properties of an object. The first thing we will need to do would be to add an object, luckily this is very simple to do.

In your **Project Panel** navigate down to **2DGamekit > Art > Sprites > Environment** and then simply find a sprite (I would recommend an Alien Statue) that you like and drag it into the Scene View. This Object is now in your game! Select this object in the **Hierarchy** and then have a look at the **Inspector**, you will notice that this inspector now has multiple components in it.

Different components do different things and you can add and remove components as you deem fit, the only required component is **Transform**. Transform is the object’s position, scale, and rotation in the world (Scene), meaning that without this the object simply wouldn’t exist.
By default your sprite should have the Transform and Sprite Renderer component. The transform does as explained above, and the sprite renderer is what’s actually displaying our object in the game. Test out the tools introduced earlier with this object and see if you can get a grasp of what each one does before continuing. Unity has a steep learning curve so if you do not grasp these concepts right away or are a bit confused do not feel discouraged, as you use the engine more you will gain a better understanding of it quite rapidly.

**Important:** Because we are working on a 2D game we will only be making use of the X and Y position, ensure that any new object you add as a Z position of 0. To stop distortion you only want to rotate along the Z axis, so ensure that the rotation for X and Y is always 0.

**Setup**

Before we can begin on our work we first need to make a new Scene for our level. The scene holds all of our objects and is where we will create the level for our players.

The package that we are using to make our game comes with a Scene Creator Tool which will make our scene and populate it with all of the required assets without us needing to do anything.

Along the menu tools at the top of the screen select Kit Tools > Create New Scene ..., this will cause a popup to appear (generally at the top left corner of your leftern most monitor) where you can specify a name for our new scene, call it Level1 and press Create.

A new scene should now be made and you should have some objects in your Scene View and Hierarchy.
The tool that we’ve just used allows us to skip a significant portion of remedial tasks in setting up a game, unfortunately it’s only available with this package so for future development using your own assets it will not be available.

**Tilemaps**

**Tilemaps** are a new feature in Unity that makes building 2D tile-based games much easier. Games like platformers and roguelikes often represent the world as a 2D grid of tiles. Laying out such levels in the normal Unity scene editor can be very tedious as you have to place each tile individually and make sure it is correctly positioned on the grid. The Tilemap editor makes this job much easier.

Generally before you are able to ‘paint’ onto a tilemap one needs to be added to your scene, luckily the Scene Creation Tool that was used earlier has already added one into the scene. Expand out the TilemapGrid object in the Hierarchy (by clicking on the triangle to the left of its name) and then select the Tilemap object. This tilemap is where all of the platforms, floor, and walls will go in the game.

To be able to ‘paint’ on a Tilemap a Tile Palette first needs to be created, luckily this is already included in the assets provided. Click on Window > Tile Palette to open up the Tile Palette window, it is recommended that after this window opens you drag it next to your Project Pane.

The provided assets have two Tile Palettes already created, swap to the TilesetGameKit as this is what we’ll be using to create our level. This Tile Palette has been set up with two Rule Tiles, what this means is that a single tile has been created and rules have been specified on which sprite to display based on what other tiles are next to it. This may sound confusing at first, but as you start to ‘paint’ the tiles it will become clearer.
The **Tile Palette** window also features its own toolbar. These tools are quite intuitive and self-explanatory. Have a play around with what each of the tools do and paint a basic level. **Tip:** Push and hold shift whilst using any tool to delete any tiles rather than add new ones.

**Important:** The players are only able to see what is within the boundaries of the **Camera** object. If there is no Camera in the scene the players will not be able to see anything. The **Camera** is the only object that should not have a Z position of 0 in a 2D game, simply because if it does the objects you want to display will not be in front of the camera. Instead it should have a Z position of -10 at all times.

In terms of making this project easier to develop it is recommended that you have no gaps where the player can fall off the edge of your world, some tips and tricks will be explained in the independent workshop to simulate this.

**Components & Prefabs**
Now is a good time to test out our game. But before we can do this a character needs to be added that the players can control, Ellen is a default character provided with these assets and she will be the protagonist for our game.

Generally when we are making a game scripts will need to be written to provide the desired control of the player, but with the package we are using one is already provided for us. What makes this easy is that Ellen is what’s known as a **Prefab**.

A **Prefab** is a special type of asset which holds all of an objects properties and components and allows the designer to easily replicate this by adding it to the **Scene**.

Normally we will need to manually add the Ellen prefab to our scene, but again the Scene Creator Tool took care of this for us.

Press the **Play** button and walk around your scene. **WASD** is to move and **Space** to jump.

What you should notice is that firstly a **User Interface** (UI) element has appeared at the top left of the screen, this represents Ellen’s health, the second thing is that Ellen constantly changes from walking to running to jumping to crouching, the third is that the game camera follows Ellen around the scene, and finally Ellen is able to collide with the walks and floor.

These all generally need to be set up manually by the game designer, but the Scene Creation Tool and Prefabs allowed all of this to be automated. **Ensure you have stopped playing you scene before you continue.**

We can see why these things happen to Ellen having a look at the components that are attached to her, select **Ellen** in the **Hierarchy** and look at the **Inspector Window**. UI will be discussed later in the class.
In order of what components are applied and what they do are:

1. **Transform**: Holds Ellen’s position, rotation, and scale in the world
2. **Sprite Renderer**: Displays Ellen’s sprite on the screen for the players to see
3. **Animator**: Controls all of Ellen’s animations, swaps from one animation to another depending on if the correct conditions are met. i.e. Is Ellen walking? If yes, display the
walking animation, Is Ellen standing? If yes, display the standing animation, etc. This behaves exactly like a state machine.

4. **Capsule Collider 2D**: Detects collisions with other objects that also have a collider, if a collision is detected it stops the objects from going through each other. As it is a Capsule Collider, it is in a Capsule shape.

5. **Rigidbody 2D**: Applies physics to Ellen. i.e. adds Gravity, Drag, Weight, Velocity, allows Collisions, etc.

6. **7 Scripts**: These are custom scripts that were written to further add functionality to Ellen, these include a basic inventory, character controls, attacks, etc.

**Important**: As Unity is both a 2D and 3D engine there are both 2D and 3D Physics Components. These are not interchangeable, there are fundamental differences in how each of these compute and if the wrong one is used it can impact your game greatly. All 2D Physics Components have 2D in the title. This only applies to Physics Components, other Components are mostly not affected between 2D and 3D.

The reason behind why Ellen doesn’t fall through the floor is a combination of the **Rigidbody** and **Collider** that is on Ellen. The **Rigidbody** is applying physics to Ellen, and hence gravity, which is causing her to fall until a collision is detected, the **Collider** detects when the object comes into contact with a secondary object that also has a collider, it registers this collision and stops the objects from moving through each other.

Select the **Tilemap** object in the **Hierarchy**. You should notice that each grid space with a platform has a green square around it, these are the boundaries of the **Collider**. When one of the edges are ‘hit’ with another **Collider** a collision is detected and the objects are halted from passing through each other.

---

**Camera**

Cameras are a very important aspect of any game because without a camera the player will not be able to see anything. A camera behaves as the player’s eyes, if one doesn’t exist the player has no visuals on what is going on inside the world.
As mentioned earlier, the camera follows Ellen as she moved around the level. This is generally a very tricky process and requires significant effort to get working correctly and smoothly. As part of the Scene Creation Tool a Camera Prefab was added to the scene, expand out the Cameras object in the Hierarchy and select the CM vcam1 object.

The MainCamera object is the one that has the actual Camera Component that acts as the player’s eyes, but the CM vcam1 object controls the movement around the scene. Press Play whilst this object is still selected and you should be able to see how this script behaves.

The yellow dot represents where the camera should be centered, the blue space is the allowed area for the camera to be centred, and the red space is the area where the camera is not allowed to be centred. Run around the scene and look at how the scripts behaves.

Whilst it’s not necessary to understand how the camera works for the development of this game, it provides a good insight into why the camera behaves as it does, and how cameras should behave in this style of game. **Ensure you have stopped playing you scene before you continue.**

### User Interface (UI)

The game is almost in a playable state, albeit a bit basic, there are just some final additions and settings to change to improve the overall ‘feel’ of our game. The first step is to add a User Interface (UI) to our game. Fortunately this has already been created for us and part has already been added to the scene. In the Project Panel navigate to 2DGameKit > Prefabs > UIPrefabs.

There is one additional UI prefab that we need to add to our scene to improve the player’s access to information. This is the KeyCanvas prefab, which allows the player’s to see how many key’s they have collected in the level. Drag this prefab from the Project Panel and drop it underneath the UI section in the Hierarchy.
When a player collects a Key throughout the level it will show up in the UI tool we just added, let’s add some keys now to ensure that this works. In the Project Panel go back up one folder to the Prefabs folder and then into the Interactables folder, locate the Key prefab and drag and drop it to the desired position in the Scene view, do this two more times.

Select one of the Keys in the Hierarchy and look at the components, more specifically the Inventory Item (Script) component.
This component controls what the object is in terms of the inventory, the **Inventory Key** property determines the unique name of the object so the game can keep track of it. By default, all of the keys are called Key1, which as you can imagine is a problem. These keys should each have a unique name of either **Key 1**, **Key 2**, or **Key 3** to properly register with the UI and Inventory script. Change the names of two keys to Key2 and Key3 in the Inventory Item component and then press **Play**.

As each key is collected the amount of Keys on the UI tracker should increase.

---

**Ensure you have stopped playing you scene before you continue.**

**Enemies & Attacks**

The package that’s being used comes with two types of enemies, a melee enemy and range enemy. Both kinds are located in the **Prefabs > Enemies** folder. These are already configured and no settings will need to be changed so they can attack the player.

Drag one of each kind of enemy into the Scene and press **Play**.

---

As Ellen gets hit a new animation should play and one heart will be removed from the UI, followed by 3 seconds immunity before she can take damage again.
Ensure you have stopped playing you scene before you continue.

Currently Ellen is not able to kill the enemies as she has no weapons, so let’s add some for her. In the Prefabs > Interactables folder there are three different weapons prefabs, they are: WeaponPickupBoth, WeaponPickupGun, and WeaponPickupStaff. These should be self explanatory for what weapons they provide to Ellen, for now just add in a single WeaponPickupBoth to the Scene and test it by pressing Play. The hotkeys to attack are: K for the Staff, and O for the Gun.

As Ellen hits one of the enemies with her Staff or Gun they will be killed.

Ensure you have stopped playing you scene before you continue.

**Interactables**

There are many different kinds of interactables that can be added to expand the scope of the level, these include: Moving Platforms, Acid Pits, Spikes, Destructible Walls, Pushable Boxes, Health Pickups, etc.

These are all included in the Prefabs > Interactables folder and most work by just dragging and dropping them into the Scene. If you would like to customise any of these interactables, speak with one of the Workshop Demonstrators for assistance.

**Decorations**

With this knowledge you should now be able to build a fully functional level, but it does look quite bare. In the Art > Sprites > Environment folder there are a large amount of decorative assets that can be added to the level, these assets offer no functionality but instead improve the overall ‘feel’ of the level.

Spend the remainder of this Workshop testing the different Interactables and Decorations or brainstorming your level for the next Workshop.
Design Tips

When designing a level there are a few tips that are often overlooked and instead bad design practices are very common. It is recommended that when designing your levels in Workshop 2 that you follow these basic tips:

- More enemies does not mean more challenge
  - Do not overwhelm the player with enemies in an attempt to improve the difficulty. Instead focus on how you can alter the battlefield to make an encounter with fewer enemies more difficult.
  - There are many alternatives to get around this, such as having a combat area over a pit of acid or spikes, or putting spitter enemies in hard to access places.
- Replicate players falling to their death with pools of Acid or Spikes
- When introducing a player to new mechanics do not overwhelm them
  - Try and introduce mechanics individually in a safe manner
  - For example: Start with a regular jump, then a high jump, give the player a weapon, have them break a wall, etc.
- Do not place Death Traps where the player can’t see
  - If you are making a ‘leap of faith’ mechanic i.e. the player can’t see the floor below the drop, do not place acid or spikes at the base
  - If you do want to have these at the base make a clearly visible safe landing place to encourage the player to go there instead.
- Do not overwhelm the player with decoration
  - If there are excess amounts of decoration on a level the player may have a difficult time either distinguishing actual level mechanics or focusing on the game itself.
  - Less is more in most cases.
- Encourage exploration
  - Linear levels do not keep players captivated
  - Create smaller sections for your level where the player has to go in multiple different directions to beat it or to access rewards e.g. health pickups.
  - Try and create a shortcut from the end of each section to the main path so the player does not have to backtrack significant portions of the level to continue