Introduction to the ECS Framework

1. Introduction

This tutorial introduces the basic framework that is provided to you. We will be looking at the framework in lectures shortly after you complete this lab. Therefore, do not worry if parts of this code look scary or confusing, as the main reason for this lab is to start to introduce you to the provided code. We will be covering the code in detail in lectures.

Included with this tutorial is example code. This acts as an initial structure that you can extend without worrying too much about the underlying Windowing system and platform.

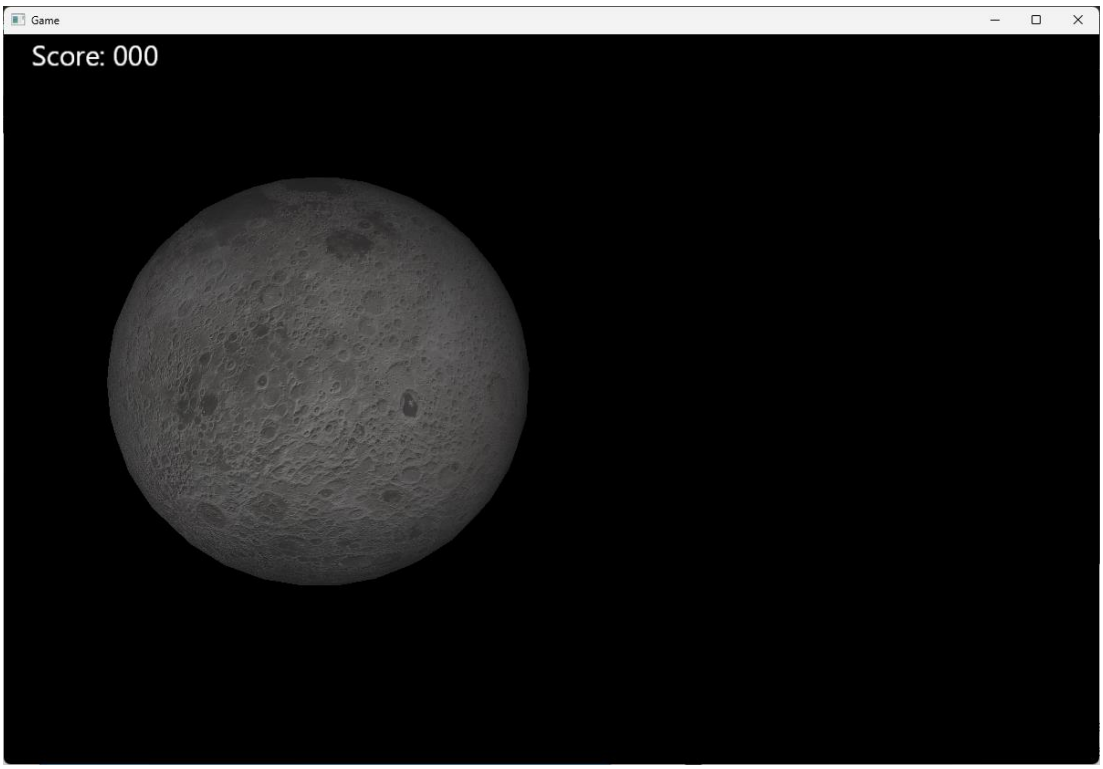
2. Setting up the Windows project

Download the accompanying file '600098-Initial_ECS_Framework.zip' and unzip it into a convenient area on your file store. Open the OpenGL_Game solution file to start Visual Studio and load the solution file.

3. Testing the project

Build and run the project to verify that you have the following.

Click the left mouse button to verify that you have the following ‘Game’.



Press the ‘M’ key to go back to the Main Menu.

4. How it works: OpenTK

One of the nice things about OpenTK is that it does much of the hard work for us, performing all the low-level Windows commands and supplying methods to do some of the basic setup. By default, it will define a number of attributes for the Window we are going to draw in. These include:

- window size and colour depth,
- z-buffer (to help with hidden surface removal),
- double-buffering (two buffers per window so that we can draw new scenes in one buffer whilst displaying the other – a standard for real-time graphics).

5. How it works: Framework

The **Main** class is the starting point of the program and creates an instance of the **SceneManager**.

The **SceneManager** class is the main game loop and controller. The **SceneManager** class inherits from the **GameWindow** class. The **GameWindow** class has a number of pre-defined methods which we can either accept or override – `OnRenderFrame()` is one such example. **GameWindow** is part of OpenTK and is responsible for the window creation etc. that we have just described. The **SceneManager** class currently creates new instances of Scene objects. Initially it creates an instance of the **MainMenuScene** scene class.

The **MainMenuScene** class is a scene object. This class currently displays the text “Main Menu”. You will want to add functionality for multiple options that can be selected by the user when you develop your game later.

Our ‘game code’ can be found in the **GameScene** class. The **GameScene** class is a scene object. This class contains our game and most of the gameplay and game logic.

The **GUI** class contains the general rendering code for drawing text in the window.

6. GameScene

Constructor

The `GameScene()` constructor is called first and is used to create instances of the game engine Managers. We will be discussing these Managers in lectures. The example code uses this constructor to:

- Adds a method to the keyboard delegate so that GameScene can choose what happened when keys are pressed,
- Set the background clear colour,
- Create the Camera (which creates the View and Projection matrices),
- Create the Entities,
- Create the Systems.

Update()

The **`Update()`** method is called before each frame is rendered. You should add any code that requires to be updated before to the scene being drawn.

This method currently sets the time taken since the previous frame (dt).

Render()

The **`Render()`** method is called each frame and it is where we render the scene. The example code uses this method to:

- Set the viewport where the graphics will be rendered,
- Clear the buffers,
- Calls the **`ActionSystems()`** method of the **`SystemManager`** (more on this later),
- Renders the score as text.

Close()

The **`Close()`** method is used to clean up the GameScene when the game has finished.

It currently removes the keyboard delegate method as GameScene no longer wants to be notified of keys being pressed.

Keyboard_KeyDown()

The **`Keyboard_KeyDown()`** method is used to handle key presses for the game.

It currently will move the camera if the cursor keys are pressed, and go back to the main menu if the ‘M’ key is pressed.

CreateEntities()

The **`CreateEntities()`** method is used to:

- Add an Entity called ‘Moon’, that has a position of (-2, 0, 0) in the scene and has moon geometry

The Moon Entity is composed of Components.
The first Component is a **`ComponentPosition`** that defines the Entity’s position in the scene.
The second Component is a **`ComponentGeometry`** that will store the geometry defined in the provided file.

The Entity is added to the List of Entities contained in the **`EntityManager`** using the **`AddEntity()`** method.

CreateSystems()

The **`CreateSystems()`** method is used to:

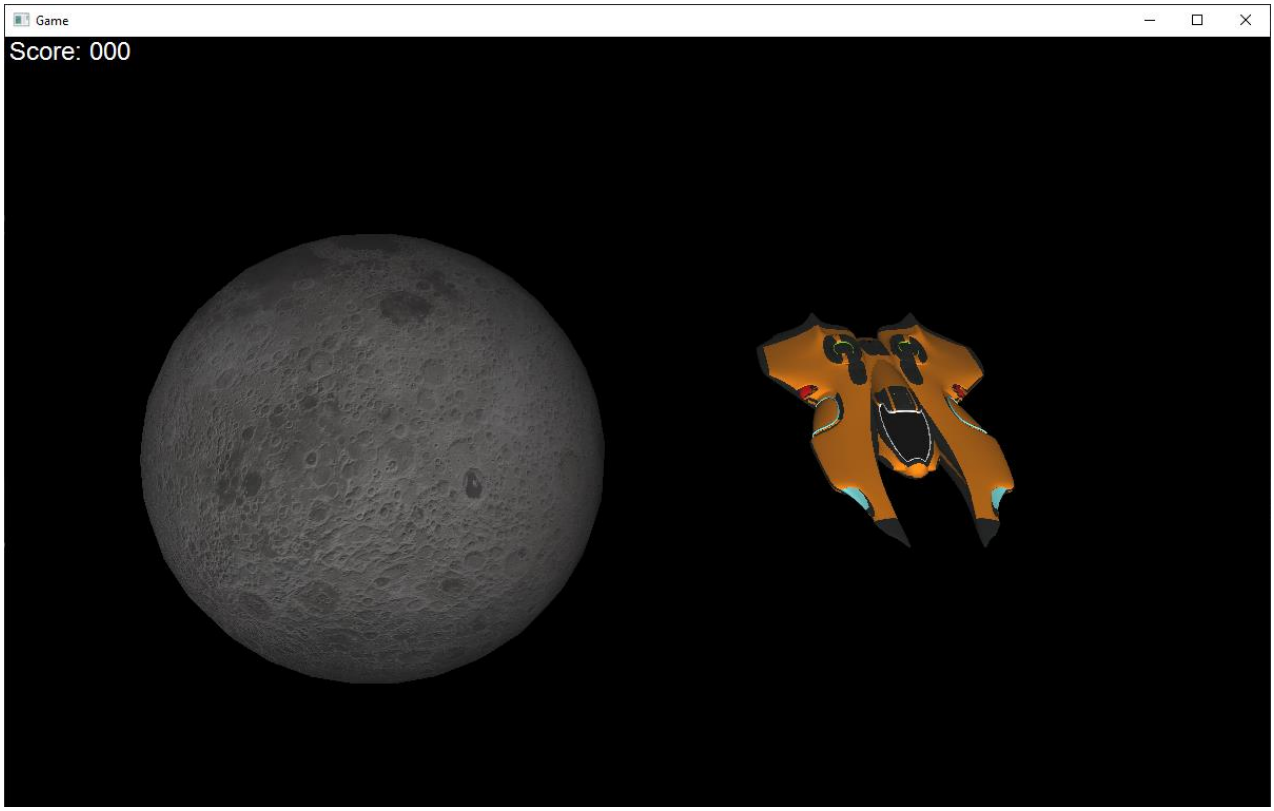
- Add a System called ‘SystemRender’, that we use to render all objects that can be drawn

Much more information on this will be discussed in lectures and future lab tutorials.

7. Exercises

Please attempt these exercises, but if you get stuck or you are confused then ask for help during the scheduled lab times.

1. Add a new **`Entity`** called ‘Wraith_Raider_Starship’ at (+2, 0, 0) with the geometry called "Geometry/Wraith_Raider_Starship/Wraith_Raider_Starship.obj". The result will be the same as below.



2. Add a new Entity called ‘Intergalactic_Spaceship’ at (0, 0, 0) with the same geometry "Geometry/Wraith_Raider_Starship/Wraith_Raider_Starship.obj". We now have two spaceships and a moon.

3. Review the code in the **`EntityManager`** and **`ResourceManager`** – we will look at the **`SystemManager`** in future lab tutorial work.

4. Review the resource files in the **`Geometry`** folder. These are the resources that allow us to render are game objects.

5. Review the code in the **`Objects`** folder, i.e. **`Entity`**.

6. Review the **`Geometry`** code in the **`OBJLoader`** folder.