COMP140-GAM160: Further Games Programmi

5: Streams & Serializatio

Learning outcomes

- **Understand** the concept of serialization in Computer Science
- **Explain** how streams can be used to send data to a location
- **Implement** a save system for games

Working with the Filesystem

- Typically in a game we don't interact with the filesystem
- We work with some sort of virtual filesystem which is local to the game
- This is because
  - We don't have access to the users root directory
  - We want to limit cheating
  - Game Engines usually access file resources in a crossplatform manner

The Filesystem & Game Engines

- Often Game Engines use a compressed format to store assets
- This aids in performance but also adds in a layer of security
- This means that if you want to load assets programmatically you can't use standard file reading functions
- See**Resource.Load** in Unity or **Referencing Assets** in Unreal

Typical Filesytem operations

- In most modern operating system you can't access the local filesystem
- You can only use certain directories such as the Users Document or AppData directory
- See **Application.*Path** in Unity **FPath** in Unreal
- These sort of functions/variables will give us access to safe directories to use
- We then use other functions to create directories and/or files
- See **File** & **Directory** class in C# or **File Management** for Unreal

Serialization

- Is the process of converting a data structure to a series of bytes
- This can be used to transmit the data structure to a file, over the network or store in memory
- We can also take serialized data and reconstruct a data structure
- This makes serialization the perfect candidate for saving game data

Streams

- Are just generic series of bytes that are sent to a resource
- This resource could be memory, a file or network
- Often used in conjunction with Serialization to save data
- See **IO Streams** in C# or **FBufferArchive** in Unreal

Exercise 1 - Basic Saving

1. Create a Player Info Screen where the player can enter a username
2. This is then saved and when the game is restarted the username is displayed on the screen
   - **Hint** Unity - use **PlayerPrefs**
   - **Hint** Unreal - search for **Saving Your Game**
3. Add in buttons to delete the saved data

Exercise 2 - Dynamic Loading

1. Create a basic scene with three cubes
2. Create three materials (red, green, blue)
3. Assign the red material to the cubes
4. On a key press load and assign the green material (you must do this dynamically) to the cubes
5. Create a Prefab/Actor of a sphere which uses the blue material
6. Dynamically Load and Spawn this sphere when a key is pressed

Exercise 2 - Hints

- Unity - Use **Resource.Load**, all assets loaded this way should be in the Resource Folder. See **Loading Resources at Runtime**
- Unreal - See **Referencing Assets** for loading assets such as materials and see **spawning actors** for spawning actors

Exercise 3 - Serialization

1. Create a directory in My Documents called **GameTitle**
2. Inside **GameTitle** create a directory called **Saves**
3. Create a scene which has a number of triggers dotted around the level
   - When the FPS Player enters one of these triggers, save the players position to the**Saves** folder, the data should be saved in a file called **save.sav**
   - If the player quits the game and restarts, you should check for the existence of this file and load the last position

Exercise 3 - Hints

- Unity - See **JSON Serialization in Unity**, this will get the data into a string. You will then need to use the **Path**, **StreamWriter** and **StreamReader** C# classes to complete the task
- Unreal - See **FBufferArchive** and **Save System, Read & Write Any Data to Compressed Binary Files**

Exercise 4 - Extension

1. Instead of saving to one file each time, create a filename with the date and time of the save. Save your game data to that
2. Create a basic UI which manages the saves (delete & load)
3. Create some basic GameObjects/Actors, these should move up and down. Save the position of these to file