It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

Working with Game Engines

COURSEWORK SPECIFICATION

It is highly advised you complete the lab work before attempting this coursework. You can re-use suitable code snippets from the lab work, but only include code that is used in the coursework. Any unused or superfluous code in your submission will be deemed as a lack of understanding the code and will result in a deduction of marks. It is advised that you make use of a source control method like Git with GitHub to track changes to your project.

This coursework will require the implementation of 2 scenes accompanied by a written report.

Mark Allocation:

- **Scene 1: 36 marks**
- **Scene 2: 40 marks**
- **Report: 24 marks**

Total: 100 marks.

SCENE 1 (36 MARKS)

In this scene you must develop functionality to load in a voxel chunk. The file format must be in XML. You should use AssessmentChunk.xml 1 & 2 as test data for loading (these are located on GCU Learn in the assignment page).

It must incorporate the following functionality:

- Loading A voxel chunk from memory.
- A first-person controller to navigate the mesh (you may use the Unity example controller as a basis).
- Place a block (with audio cue).
- Destroy a block (with audio cue).
- Instantiate dropped collectable block when block is destroyed.
- Physics should be applied to the dropped blocks.
- Keep an inventory of dropped blocks.
- Sort Inventory (option to sort by name or number held).

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

- Search Through inventory (by name)

LOADING (6 MARKS)

Your program should be able to load a chunk from an XML file. For full marks, incorporate a text entry UI element that allows the user to type in the file name to load. Otherwise you should load AssessmentChunk1.xml as the default file name. You can assume the chunk is a 16x16x16 3D array. You should observe the structure given in AssessmentChunk1.xml.

EVENTS (8 MARKS)

You should maintain modularity in your code using events. This should include destroying blocks, placing blocks and any audio involved. For full marks, play different audio to represent the different block types (for both destroying and picking up).

DYNAMIC MESH (6 MARKS)

Use the texture supplied in the lab exercises. You must be able to destroy blocks on your mesh and be able to place blocks of any type (you will be using the four types used in the lab - grass, dirt, sand and stone). You must implement a mechanism to switch between these types.

INVENTORY (8 MARKS)

You must incorporate a mechanism for creating dropped blocks. Each block must be textured to represent the type of block that has been destroyed. These blocks should be collectible and you should keep an inventory of blocks that have been collected and provide basic UI to view this inventory (player character movement and player character interaction should be disabled **if** this is a new screen that appears with a button press). You should provide a mechanism to sort this inventory using **merge sort** based on the amount of each item you have and by the name.

For full marks implement **ONE** of the following features:

- Build and texture the mesh for the dropped blocks in code.

Or

- Allow blocks to be drawn towards the player by applying force on a button press.

SORTING (8 MARKS)

You must implement **merge** sort. For full marks you must provide a way to sort by name (A to Z and Z to A) and amount (low to high and high to low) **Hint**: Use delegation. This should be reflected by the UI provided for the inventory.

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

SCENE 2 (40 MARKS)

For the second scene, you must develop functionality that allows a 2D platformer character to enter a trigger volume and engage in a dialogue system in which the player selects options based on what the non-player character (NPC) said. The Dialogue must be stored in XML and loaded into the scene. A Hypothetical developer must be able to load and save this dialogue using a custom editor window.

Functionality Required:

- 2D follow camera.
- A Dialogue Editor.
- Save Dialogue to XML.
- Load Dialogue from XML.
- Dialog system that triggers when the player gets to the non-player character.
  - Should be interactable by the player through a GUI.
  - The dialogue that plays should be selectable in the inspector (i.e. the developer types in the name of the file).
  - Be wary about redundant code, think about what can be re-used across the dialogue system, editor and the saving/loading code for both.

A package (**WGE 2019 Assignment Scene 2.unitypackage**) containing the skeleton scene with the 2d character, scene geometry and NPC placement is provided in the Assignment section of GCU Learn and should be used as a basis for Scene 2.

DIALOGUE SYSTEM (6 MARKS)

The scene requires a dialogue system where a hypothetical NPC speaks and the player can pick a response which will lead to the NPC saying something different based on the player's response. The dialogue spoken by the NPC can be text displayed in an appropriate GUI and the player's options can be buttons that are also displayed on that GUI (you may pull inspiration from games like banner saga or dragon age as to how the GUI should be laid out).

The player cannot move the character when the dialogue sequence triggers but can move the player character again once the dialogue sequence ends.

The dialogue sequence should trigger when the player get's close to the NPC character (the red rectangle pre-placed in the level.)

DIALOGUE EDITOR (10 MARKS)

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

Making use of Unity's editor window API, you should construct an editor window that allows you to build a dialogue sequence for the dialogue system described previously. It must be able to:

- Allow the developer to create an arbitrary amount of exchanges between one character and the player.
- The developer can define an arbitrary amount of responses the player can chose from.
- The developer can define what the next line of dialogue spoken by NPC will be based on what option the player selected.
- The developer has the option to save the dialogue sequence into XML.
- The developer has the option to load the dialogue sequence from XML.
- GUI layout should be clear and easy to follow.

To test that the dialogue being produced by this dialogue editor is functional, the following dialogue sequence in figure 1 should be created using the editor created in the student's project:

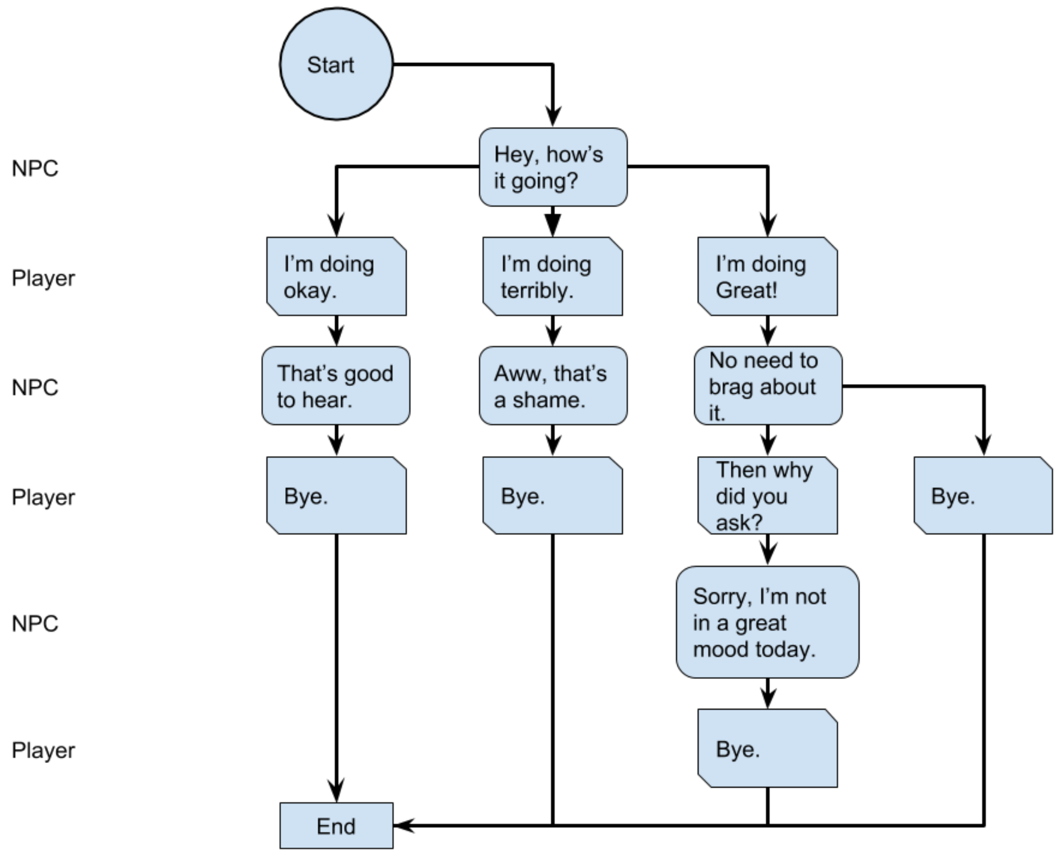

Figure 1: Example dialogue tree, main test case.

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

Note: While the tutor will also be using this test case to test the dialogue editor, this conversation should be created by the student and placed in the scene for the built version of the project.

SAVING (8 MARKS)

After the dialogue is created, the editor should be able to save the dialogue into an XML file, this requires that you think about how the XML should be laid out.

LOADING (6 MARKS)

The dialogue editor and dialogue system need to be able to load the created conversation from XML into the scene so that it can either be further edited or used in gameplay.

2D CAMERA (10 MARKS)

The 2d premade character needs to navigate the level and the camera must be able to follow them in a way that:

- Keeps the character on-screen.
- Is pleasing to the player.
- Can change it’s focus onto other game objects that are important when necessary.
    - For example, the camera should switch its focus to the NPC at the start of the dialogue sequence.
- The camera should shake momentarily when the player lands on their feet.

The camera should not be attached to the character as a child component and sensible use of events and coroutines is expected. The camera should be appropriate for the scene and care and attention should be brought to making it feel good for the player.

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

REPORT (24 MARKS)

The report should contain a front page clearly stating **name, matriculation number, course, GitHub Repo link** and the following disclaimer:

> *I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.*
>
> *Signature.*

This report should be titled:

> *Working with Game Engines Coursework: Report*

The report should have either 11 or 12pt. font and should be formatted sensibly with use of annotated code fragments where appropriate with the **maximum** word count being 3000.

Failure to follow these requirements may result in a loss of marks for the report.

SCENE 1 (8 MARKS)

Give an overview of the scripts used to implement this scene. This should include explanation for important methods and variables used. You should make special note of how **events** are used in the scene and the **communication between scripts**. Describe what **software design patterns** have been used, and how these are implemented including the script or scripts involved. You should also discuss any sorting algorithms used and their operation.

SCENE 2: PART I (6 MARKS)

For this scene, aim to describe what **techniques** and **software design patterns** you used to control the camera and make it follow the player. Have a look at ***PlayerController2D.cs*** and ***PlayerMovement2D.cs*** used for the 2d platformer character, can you describe what **techniques** and **software design patterns** were used and why they are suitable (or if you even think it's suitable)?

SCENE 2: PART II (6 MARKS)

Describe the structure of the dialogue file you loaded in the scene. You should use a diagram to describe the DOM for the file. You should also describe the dialogue editor, it should read like a tutorial which explains to the reader (i.e. a hypothetical developer) how to use it. This will be judged in how easy it is to follow for the tutor for when they try to re-create the conversation shown in figure 1.

FORMATTING AND AESTHETIC (4)

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

The report should make use of sensible formatting with code fragments and images clearly annotated. There should be sensible headings for sections and sub sections with a table of contents at the beginning of the report.

Hand in details

The final build should have a menu scene to select between the 2 required scenes. A simple scene with 2 buttons is sufficient. The build should be for Windows machines.

The Coursework should be submitted via **GCU Learn** by **5pm Friday 26th April** on Week 13. Your project file and report should be compressed into one zip file (the file format must be **.zip NOT** .rar or .7z). The zip file should have the following name format:

<Last Name> _<First Name>_ WorkingWithGameEngines.zip

Your zip file size should **not exceed 400MB** and **MUST** include the following:

- **The build data folder with the executable.**
- **The project folder**
- **A .docx or .pdf file report (name this *<Last Name>_ <First Name> _WGE Report2019*)**

Marks will be deducted if any of these items are missing or incomplete.

It is advised that students allocate sufficient time to complete this course work. Completing lab exercises will be necessary to acquire the knowledge and practice to implement a solution.

APPENDIX: MARKS BREAKDOWN

SECTION	MARKS (UP TO)
SCENE 1	***36***
Loading	6
Events	8
Dynamic Mesh	6
Inventory	8
Sorting	8
SCENE 2	***40***
Dialogue System	6
Dialogue Editor	10
Saving	8
Loading	6
2D Camera	10
REPORT	***24***
Scene 1	8
Scene 2: Part i	8
Scene 2: Part ii	8
TOTAL	***100***