**Portfolio Part One (UFCFJL-30-1)**

**Student Name: Jamie Serlin**

**Student Number: 24030960**

**Glossary**

* Add each new term you discover throughout the module along with a definition.
* Higher grades will be awarded for students who also include a reference/citation to the location of the definition.
* **Estimated 1-2 pages**

|  |  |
| --- | --- |
| Normal | The direction that a face is facing on a mesh, perpendicular to the tangent |
| Subsurface scattering | A graphical technique that approximates light passing through a translucent surface such as hands or leaves |
| UV Unwrapping | The process of folding out a 3D model to allow textures to be applied |
| PBR | Physically Based Rendering is a rendering technique that uses additional textures for effects such as roughness, metalness, and extra normal detail with normal maps |
| High Concept | A short summary of the main design features and ideas of a game. |
| Vertex | A single point in the world, with x,y,z coordinates (and extra data for UV mapping). It is used in 3D modelling as they can be connected with edges and faces to create a mesh. |
| Linked List | A data structure in which multiple elements contain a reference to the next item of the list. This makes it easy to insert and delete elements. |
| Big O Notation | A mathematical concept to describe how fast different sorting algorithms work depending on the size of the set. |
|  |  |
|  |  |

**Task 1: Tools and Techniques**

Choose **three** game engines to compare in the table below. Include the following:

* Pros – Advantages of the engine
* Cons – Disadvantages of the engine
* Appropriate Usage – Situations/projects where this engine would be most appropriate to use

Higher marks will be awarded for students who look beyond materials taught in class.

* **Estimated 1-2 pages**

|  |  |  |  |
| --- | --- | --- | --- |
| **Game Engine** | **Pros** | **Cons** | **Appropriate Usage** |
| Unity | -Easy to understand the interface and get started  -Many packages and in-built components  -Probuilder tool lets you create map blockouts in engine very quickly  -You can make your own in-editor tools using C# and Unity’s API  -Works very well in both 2D and 3D | -Can have a lot of slowdown when projects get large  -Not many high-quality assets on the asset store | -Great for coding experiments, as well as indie games.  -However, it is not well suited to large or AAA games as it lacks robustness. |
| Unreal Engine 5 | -Extremely powerful graphics engine, with tools such as nanite and lumen for real-time global illumination.  -Blueprint nodes let you script gameplay without needing to learn a language such as C++  - Quixel Megascans provides a huge library of high quality photoscanned assets to use for free in the engine. | - Quite demanding on performance, meaning you could be cutting out a large part of your playerbase  -Not very good at making 2D games as it was designed primarily for 3D. | Unreal can be used for just about anything, with many large AAA titles being on this engine (Fortnite, Silent Hill 2), as well as smaller indie games (Grey Zone Warfare, Hi-Fi Rush).  -Can also be used for films, architecture visualization and much more. |
| GameMaker 2 | - Specifically tailored to 2D games, so has lots of support for things like tilemaps and spritesheets.  - Very fast opening and build times, meaning it is quick to iterate on code.  - Simple and intuitive interface | - Custom scripting language means that there is not a lot of outside libraries, and you will need to write a lot of your own methods.  - No native video-playing support  - £300 to buy, whereas other engines are free to use. | - Great for any 2D games  - Good for beginner devs, especially those new to coding |

**Task 2: Algorithms**

Fill in the Big O Complexity Chart Below:

Choose **three** sorting algorithms and list their best case and worst case complexity in the table below:

|  |  |  |
| --- | --- | --- |
| **Sorting Algorithm** | **Best Case Complexity** | **Worst Case Complexity** |
| Insertion Sort | O(n) | O(n2) |
| Merge Sort | O(n log(n)) | O(n log(n)) |
| Bogo Sort | O(n) | O(1) |

* **Estimated 1-2 pages.**

**Task 3: Data Structures**

From the provided flow chart below, define the following data structures, and give an example of what would be an appropriate use:

* 2 Primitive data structures
* 2 Non-primitive linear data structures
* 1 Non-primitive non-linear data structures

* **Estimated 1-2 pages.**



-Integer

An Integer is a whole number, meaning it cannot have a decimal place. It has one bit reserved for the sign (if the number is positive or negative), and the rest store the value. Typically, an integer in C++ is 32bit, meaning it can count from -2147483648 to 2147483647.

It is used in many different circumstances, for example counting how many times a player has died, or an ammo counter for a gun, as you cannot have half of a bullet.

-Character

A Character is essentially an int, however each value is tied to an ASCII character such as ‘G’ or ‘m.’ Making an array of characters allows you to store entire words or sentences (usually called a string).

They are very useful for things like storing chat messages online, subtitles and any text in a game such as the main menu screen.

-Array

An array contains multiple values of data of a specific type in a single variable. It has a fixed size, meaning you cannot add or remove an element from the array (without creating a new one). However, you can modify the variables inside the array by indexing it with an integer. The first element is 0, the next element is 1 and so on.

An example of a use could be a hotbar feature similar to Minecraft or a leaderboard in a competitive game.

-Linked List

A linked list is similar to an array, however it gives you much more control over the order and size of the structure. It is made up of a HEAD and multiple Nodes that store data as well as the pointer to the next Node of the list. The final Node points to NULL. This is useful for something like a list of enemies that are alive, that has to be dynamically changed in size and content.

-Tree

A binary tree is a hierarchical data structure where each Node can have up to two connecting nodes. It starts at the Root, and the ends of the tree are called Leaves. Each Node contains the data that it stores, as well as two pointers, one for the left branch and one for the right branch. The left or right can be set to nullptr, which means there is no branch coming from that side.

They are a hierarchical structure, so are very efficient for searching for data (e.g. "find all users that signed up between 2020 and 2021").

**Task 4: AI in Games**

Choose 2 algorithms and visualise the order of path planning using a flow chart.

Higher marks will be awarded for students who look into path planning algorithms beyond what is taught in session.

* **Estimated 1-2 pages.**

-Breadth-First Search



**Task 5: Rendering**

Using the shader provided at the link below, make at least one change. Provide a screenshot of the resulting shader, a screenshot of the changed code, and a brief description of what you changed.

<https://www.shadertoy.com/view/WtdSDs>

The below are the descriptions for each of the variables. An example layout for this task is provided on the next page.



* **Estimated 1-5 pages**
* **Example:**





This is the original project and therefore no changes have been made. This is an example layout for what we expect to see for this task.