H446 OCR Programming Project

*Eshan Fadi*

*Upton Court Grammar School*

*Centre Number: 51427*

*Candidate Number: 8052*

[Project Name to be decided]

Table of Contents

[Project Ideas 4](#_Toc171375809)

[Engine and Language Evaluations 5](#_Toc171375810)

[Unity – C# 5](#_Toc171375811)

[Unreal Engine – C++ 5](#_Toc171375812)

[Godot Engine – GDScript 5](#_Toc171375813)

[Python (using Pygame) 5](#_Toc171375814)

[RPG in a Box – Bauxite 5](#_Toc171375815)

[Chosen Idea and Justification 6](#_Toc171375816)

[Analysis 6](#_Toc171375817)

[Problem Identification 6](#_Toc171375818)

[Stakeholders 6](#_Toc171375819)

[Research 6](#_Toc171375820)

[Titanfall 2 6](#_Toc171375821)

[Doom (2016) and Doom Eternal 6](#_Toc171375822)

[Half Life (and other Source Engine games) 7](#_Toc171375823)

[Essential Features 7](#_Toc171375824)

[Movement 7](#_Toc171375825)

[Combat 7](#_Toc171375826)

[Sound Effects 7](#_Toc171375827)

[Multiple Levels 7](#_Toc171375828)

[Sanity 7](#_Toc171375829)

[Scoring System 8](#_Toc171375830)

[Merit 9](#_Toc171375831)

[Death Screens 9](#_Toc171375832)

[Graphical User Interface (GUI) 9](#_Toc171375833)

[Limitations 9](#_Toc171375834)

[Solution Requirements 9](#_Toc171375835)

[Success Criteria 9](#_Toc171375836)

[Design 9](#_Toc171375837)

[Decomposition 9](#_Toc171375838)

[Structure Definition 10](#_Toc171375839)

[Algorithms 10](#_Toc171375840)

[Usability 10](#_Toc171375841)

[Variables 10](#_Toc171375842)

[Data Structures 10](#_Toc171375843)

[Classes 10](#_Toc171375844)

[Validation 10](#_Toc171375845)

[Iterative Tests 10](#_Toc171375846)

[Post Tests 10](#_Toc171375847)

[Development 11](#_Toc171375848)

[Evidence 11](#_Toc171375849)

[Prototyping 11](#_Toc171375850)

[Modules 11](#_Toc171375851)

[Comments 11](#_Toc171375852)

[Naming 11](#_Toc171375853)

[Validation 11](#_Toc171375854)

[Reviews 11](#_Toc171375855)

[Development Testing 11](#_Toc171375856)

[Evidence 11](#_Toc171375857)

[Fails 11](#_Toc171375858)

[Evaluative Testing 11](#_Toc171375859)

[Evidence 11](#_Toc171375860)

[Usability Tests 11](#_Toc171375861)

[Evaluation 12](#_Toc171375862)

[Success Criteria 12](#_Toc171375863)

[Usability Features 12](#_Toc171375864)

[Limitations 12](#_Toc171375865)

[Maintenance 12](#_Toc171375866)

[Bibliography 12](#_Toc171375867)

# Project Ideas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Idea | Short Explanation | Complexity | Language | Computational Methods |
| Movement first-person-shooter game | A game where the player must simultaneously manage their momentum and there aim | Medium - ample use of math and GUI. Movement must be fine-tuned along with movement based bug fixes. Player physics, and bullet physics, enemy AI, and modular interconnected components, OOP, libraries | C#, C++, GDScript, Python, easier with engines | OOP, database, modular design, abstraction for the GUI, reusability for functions |
| Game engine | An application designed to help and aid someone in the creation of game or other app | Very difficult - lots of math, OOP, GUI, implementing a physics engine, along with functionality for any component, must include compilers and built in libraries to assist in building the game | C#, C++ | OOP, decomposition of functions |
| Programming Language | Writing a language and run it either being compiled, or to interpreted | Easy - filled with mainly trees and if statements, and exception raising, must have built in functions, handle memory and add basic data structures | C#, C++, CPython | Selection, trees, decomposition and modular design for different parts of the compiler |

## Engine and Language Evaluations

Engines are designed to speed up the development speed. Usually, these contain tools that handle significant background processes, such as the script execution order, a built in physics engine, and a lighting pipeline. Many moderns feature a modular design for their workflow, with there being objects which you can attach components to, to add further functionality to the object. Some of these may be colliders, materials, or even scripts.

### Unity – C#

Released in 2005, Unity is an extremely useful engine that has great capabilities for both 3D and 2D, with powerful tools to assist in the creation of apps (mostly games) and experiences. It has a huge community surrounding it, making it incredibly easy to learn, with a wide variety of plugins to aid in the development of certain aspects of the process. The currently supported language used within it C#, part of the C family. The language is very easy to learn, having aspects similar to Python and Javascript. It also simultaneously has the speed and access of C++, as it is a compiled language. I have some previous experience in using both Unity and C#, but the majority of their functionality is still to be explored. However, Unity recently made some economic decisions that have severely damaged the reputation of the company and software. As such, if I wish to continue the project afterwards, I may end up having to completely transfer the entire project to another piece of software, so this may not be the best long term solution.

### Unreal Engine – C++

All the way from 1998, this engine is well known for boasting incredible visuals without much effort required from the developer. It is comparatively easier to optimise the application to run faster, as certain mesh optimizations exist natively to Unreal Engine 5 (namely Nanite). Proven its ability time and time again, the Unity scandal has brought more eyes upon this amazing piece of software. However, due to its fabulous visual capabilities by default, making lightweight applications and games is far tougher, as many layers must be stripped away. Furthermore, it is well known that Unreal Engine is not built for the 2D experience, limiting the type of content that it can create. The language of choice for UE5 is C++, a language notorious for being extremely difficult to learn with low-level concepts such as pointers mixed in with high-level abstractions such as object-oriented programming. I have little-to-none prior experience programming with C++, so the learning curve to use UE5 is far greater than the other options.

### Godot Engine – GDScript

Godot is the game engine equivalent to Blender 3D in the 3D modelling world - a completely free and open-source alternative, that is only getting better and better every update, and will most likely become the industry standard within a decade. It uses a node-based system to construct the world and all objects which is similar to the systems that other game engines use. This intuitive system allows for more flexibility, greatly helping in the learning process. Godot mainly uses GDScript, a custom language which the engine supports completely for all platforms. It also supports C# and C++, but C# is not currently available to be compiled for the web in Godot 4.2 (the latest version of Godot at the time of writing). Godot can also support any other language using extensions to the engine. The worthwhile option would be GDScript, which is similar in syntax to Javascript and Python, and has many built in functions such as is\_on\_floor() to handle common and simple checks automatically. The amount of tutorials that the community and the Godot team themselves have made is vast and is only increasing, as many people flock from Unity to Godot, making it so that learning the engine and its language would be far easier than before.

### Python (using Pygame)

Python and Pygame together allow anyone to easily pick up and create anything and everything that they want. Unlike the other entries on this list, this is not an engine, which means that the creator must program in every feature, from collision handling and response, to manually testing individual locations for objects, instead of placing them in a visual scene and moving them easily. This significantly slows down the development of the application, and can be extremely tedious as everything must be done by the developer, reducing the amount of creativity and passion that can be put into the project. While the language itself is very easy to learn, the underlying concepts behind game development such as matrix transformations and quaternions are extremely difficult to understand and waste one’s time when implementing them in such a small timescale. Such a project would be better suited to one with a far longer time frame to work within.

### RPG in a Box – Bauxite

Made using Godot, RPG in a Box is a game engine that has everything built within it for you: a voxel editor, fully working dialogue system, a sound effects generator, and even a visual scripting language. While many of the other engines on this list also have visual scripting as part of their arsenal, none use it or recommend it as much as RPG in a Box. It also has a secondary fully working scripting language called Bauxite, which is extremely similar to Lua, a famous programming language that has been used to make amazing games such as Hades. The engine is designed to be as friendly to a normal person as possible, such that no prior experience is required to even begin making something within this engine, but this also happens to be its downfall. Since so many features are required to use the built in system to have full functionality, it becomes harder to implement higher level concepts without effectively writing your own engine.

## Chosen Idea and Justification

The game engine is far too complex and time consuming for the constraint of time, while also requiring more learning than either of the other options. The programming language on the other hand, is more simple and would not allow me to easily incorporate as many computational methods. The movement first-person-shooter game is a great medium between the complexity of the two other options, while also allowing me to incorporate a great deal of creativity and interesting implementations of the course content. Furthermore, I will be using Unity to implement this since I have prior experience using the engine and C# as well, so I can spend more time focusing on constructing the project than learning how to use the system itself.

Analysis

## Problem Identification

An up and coming group of the gaming community is a collection of speedrunners, craving for fast movement and fluidity to beat the games faster and faster. To satiate this need, I have decide to construct a fast-paced movement first-person-shooter game, demanding both an understanding of momentum and precise aim, to hit a selection of enemies and reach the end of the level as fast as possible. The game will also have a secondary genre of a role-playing game, allowing players to progress and unlock more movement abilities and upgrade already existing ones to navigate the game at faster rates.

## Stakeholders

DrunkDriving ™ - Amarveer Flora (CEO)

Silly Australian ® - Henry Masters (Founder and CEO)

## Research

### Titanfall 2

Released on 28th October, 2016 by Respawn Entertainment and published by Electronic Arts, Titanfall 2 is movement shooter, where the player swaps between moving around as a human character called a pilot and playing within a Titan, where the style of play completely changes. As a human, the player has access to wallrunnning, sliding, double jumping, and other abilities depending on the level. As a Titan, the player is able to gain great combat abilities, specific to the kit that they choose to apply to the Titan, all of which greatly overpowers a human in every aspect. However, the movement capabilities of the Titan are greatly reduced, leading to the only shared ability between all Titan kits to be the ability to move around. Some kits can fly, others can dash, but most of them have no extra movement capabilities. The 20-foot tall war machines also struggle to get in some places, and as such the player will need to frequently exit them to reach other places, or remove large volumes of health from enemy Titans via hijacking. This excellently illustrates the design philosophy for each of the levels in Titanfall 2 – ‘211’: 2 parts pilot combat, 1 part pilot movement/puzzle solving, and 1 part Titan combat. Each level was designed to focus around a different feature. For example, in the level Cause and Effect, the player swaps between the past and the present, managing enemies in both timelines, and using the changes between the two to help in accessing different areas.

### Doom (2016) and Doom Eternal

### Half Life (and other Source Engine games)

## Essential Features

### Movement

Walking

Sprinting

Jumping

Crouching (both toggle and hold)

Sliding

Wallrunning

Grappling

Double Jump

Boosting

### Combat

Fire

Zoom

Swap weapon

Quick melee

Pickup weapon

Reload

### Sound Effects

### Multiple Levels

### Sanity

The sanity meter is a measure of how aligned to the default programming the player is. Pacifism goes against the original programming: this decreases the sanity meter. Killing follows the original programming: this increases sanity meter. Secrets decrease the sanity meter, as they reveal the true nature of things. A lower sanity meter would represent the player character questioning the morality of what they've always been taught and programmed to do. It will cause:

* Decrease in health regeneration speed
* Increase in damage taken
* Player will see hallucinations
  + Grants the ability to attack friendlies
  + Enemy civilians may be on your side
  + Relates to PTSD type event at the beginning of the game – the only cutscene of the game

A higher sanity meter represents the player character believes what they have been taught and programmed is the right thing, leading to:

* Increase in health regeneration speed
* Decrease in damage taken

This works with the moral of the game - doing the right thing is almost always harder than doing the wrong thing, and sometimes prejudice will perpetuate persecution.

### Scoring System

Each level is scored based on:

* Time to complete
* Enemies killed
* Secrets collected
* Inverse of death count

Further point bonuses will be granted for:

* Using less weapons
* No deaths
* Pacifist (only applied on the final level if the player has killed 0 enemies across the whole save)

This should encourage players to return to the level and get a higher score, potentially seeing the flower planting system. The score will be displayed in a room, and the player can interact with each of the elements to show the score for each category

### Merit

The currency of the game is called merit. It can be obtained by picking it up from within the level (such as the bodies of friendly soldiers) and also from the score at the end of the level. Killing more and more difficult enemies will lead to more merit, although to stop merit-farming tactics, it will be limited to this equation:

.

Merit will allow the player to buy upgrades for their player character, and unlock more weaponry/ammo types to use in the pre-level inventory setup. However, ability unlocks will be based upon the level progress, since it will be tied to the story.

### Death Screens

### Graphical User Interface (GUI)

## Limitations

## Solution Requirements

## Success Criteria

# Design

## Decomposition

## Structure Definition

## Algorithms

## Usability

## Variables

## Data Structures

## Classes

## Validation

## Iterative Tests

## Post Tests

# Development

## Evidence

## Prototyping

## Modules

## Comments

## Naming

## Validation

## Reviews

# Development Testing

## Evidence

## Fails

# Evaluative Testing

## Evidence

## Usability Tests

# Evaluation

## Success Criteria

## Usability Features

## Limitations

## Maintenance

# Bibliography

Anon., 2024. [Online]   
Available at: https://tf2sr.github.io/TF2SR-Wiki/