**COMP3567 Game Specification Form Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| **Marking Criteria** | **Describe how your game matches the criteria (Description of each item is limited to 50 words)** |
| **Game design matching the ‘Covid-19 Fighter in the UK’ theme (5%)** | |
| Justification of the choice of game type: | I’ve played games for many years and some of my fondest memories have been playing movement-based first person shooters such as Team Fortress 2 or Doom. I’ve decided to make a fast-paced platformer fps in this style. |
| Game story: | You are a newly built memory B cell in a body that has just been infected with COVID-19! Use your blaster and grappling hook to navigate the bodies veins while fending off any unwanted pathogens that stand in your path.  As the game progresses, the levels become harder to navigate and the enemies become more numerous as the infection gets worse. In the final level, even the friendly T cells have turned against you in a cytokine storm! You’ll need your wits about you if you’re going to beat this. |
| **Core development and implementation (30%)** | |
| Game scene (visual representation [2D, 2.5D or 3D], internal data structure): | The scene is a Unity3D game scene where objects are stored as vector arrays by the Unity back end. I constructed the more complex objects using the Unity ProBuilder package.  For the COVID cells’ pathfinding (described in more detail in the “Advanced interaction” section), each flying nav mesh is a 3D array of voxel objects, where a voxel is a Boolean stating whether or not a terrain object exists within the voxel. |
| Game flow and how it is designed (e.g., navigation, screen scrolling, levels): | The game consists of 4 levels of increasing difficult and stakes. Each individual level can contain any number of checkpoints. When a player enters the invisible bounding box of a checkpoint this becomes their respawn point (unless the checkpoint is less advanced than their current one meaning they have backtracked through the level). When the player dies, they respawn at their respawn point.  Each level can also contain a number of “battle arenas”. Each battle arena can have a number of enemies attached to it, when a player enters a battle arena they too become attached to it. Once a player becomes attached to a battle arena, all enemies attached to it spawn. The player remains attached to the battle arena until all enemies attached to it are dead, at which point the battle arena’s termination code runs (the level ends, a door opens, etc.). If a player dies while attached to a battle arena, all enemies attached to it respawn (and their health is reset to full).  The general layout of a level consists of a number of battle arenas separated by parkour sections. Although, later in the game (particularly the third level) battle arenas and parkour sections are mixed, requiring the player to use all the skills they’ve learned simultaneously if they’re to stay alive. |
| Game interaction (e.g., action detection and response generation): |  |
| Game object (e.g., use of sprite, 3D objects, animation, multimedia): | As mentioned above, the game objects consist of vertex arrays with some fairly complex terrain objects (unity terrain, stairs, arches, doorways, etc.). Beyond this, I also used some models I found online for bacteriophages and coronaviruses. I also imported some animations for these enemies which I modified to match the speed and intensity of the game. Beyond this, I made some of my own animations for the player and UI elements.  I also imported some S3TC textures from the unity asset store which I modified in photoshop so they look like the inside of body. These textures came with normal and bump maps which also required modification and provide a sense of depth and realism to the textures in game.  Furthermore, I imported the unity particle effects package from the unity asset store. This provided a set of generic particle effects which I combined and modified to suit my needs. I developed particle effect for the big walker’s poison smoke from scratch myself. This required an understanding of unity’s animation system as well as the theory behind billboarding and sprite generation. |
| **Game mechanics development and implementation (30%)** | |
| Main game rules / logics to control game progression, difficulty and end game conditions: | Machinations diagram is required for this item on top of the description. |
| Control of game object abilities: | Machinations diagram is required for this item on top of the description. |
| **Good use of game engine (12%)** | |
| Justification of the choice of game engine (pyGame, Unity) in terms of suitability of matching the theme and the expected target audience (game player): | Given the first-person nature of the game and the complex physics-based platforming, the most logical choice was Unity as it’s physics engine in more advanced and it’s simple to develop 3D games. |
| Types of user input supported (keyboard, mouse, joystick, etc.): | /\* I think as I use generic GetKey(“horizontal”) type stuff, it will support more than mouse and keyboard. Will need to check though\*/ |
| Types of game object interaction supported (e.g., event triggering, collision detection): |  |
| Other game engine features used (e.g., asset, incorporation of external libraries): |  |
| **Demonstrate creativity (15%)** | |
| Effective use of multimedia content: |  |
| Advanced interaction implemented (e.g., game physics, object tracking, steering behaviour): | All movement of the player and enemies is based around physics, using character controllers that I coded from scratch. This allows for complex interactions between the player and other enemies, one such example of this is rocket jumping. The main mobility tool the player has access to is their rocket launcher. Shooting it at enemies will push them around and shooting the floor beneath your feet will propel you into the air, chaining rocket jumps together can result in quick movement over complex terrain. Beyond rocket jumping the player can use dashes which impart a force on them and a grapple hook which generates a spring connection between the player and another game object, chaining these moves together results in a huge amount of mobility for the player.  For the COVID cells I wanted them to fly in an organic and dynamic way. However, the Unity navmesh only supports agents traversing 2D terrain. To get around this, I implemented a hierarchical A\* search where the game scene is divided into different 3D navmeshes through which COVID cells navigate until they reach the player, I also added a heuristic where a path that can see the player (using raycasting) will immediately terminate the algorithm as from there the COVID cell can fly straight towards the player, although this can result in a sub-optimal path being used it does find a fairly good path and terminates much quicker than the standard algorithm which was necessary as 3D A\* is computationally intensive (even with the hierarchical improvement).  To make the COVID cells’ routing more natural, I also implemented collision avoidance and steering behaviour.  All enemies that shoot (including T cells) implement object tracking and detection. In particular, the T cells efficiently locate a nearby enemy to which an unobstructed vector can be generated. This is done using physics check spheres for efficient collision detection and raycasting to determine if the object can be seen directly. |

**\*Note:** Your work must be done by yourself and comply with the university rules about plagiarism and collusion. (https://www.dur.ac.uk/learningandteaching.handbook/6/2/4/)