**Games Engine Construction Report**

**User Guide**

The demo is a simple platforming game where you have to survive for as long as possible while platforms scroll downwards and spikey balls attack you. Your score is equal to the number of seconds you have survived. My current high score is 63, if you want to try and beat it.

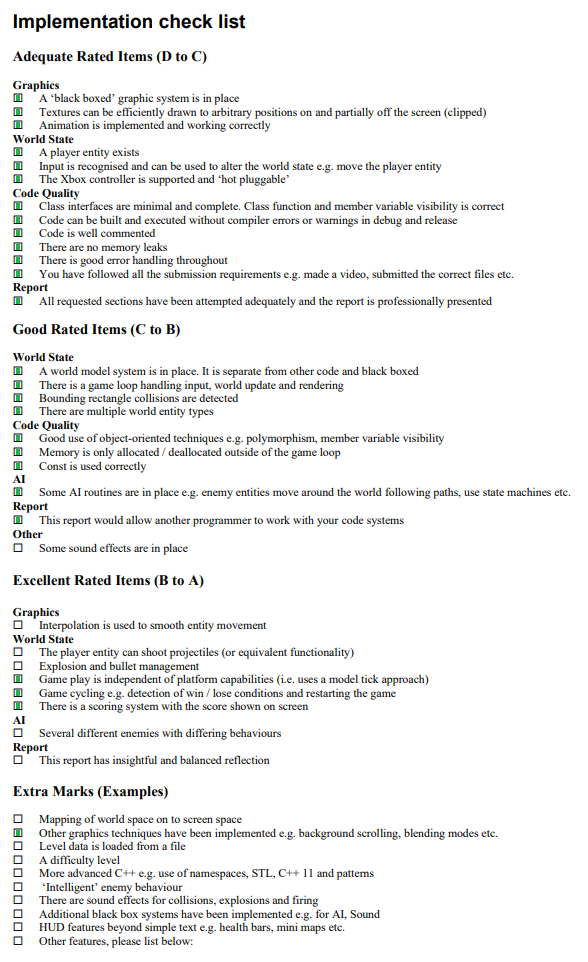
The controls of this demo is relatively simple. You can use the arrow keys on the keyboard, with left and right moving you left and right, and up being the jump button. Alternatively, you can change the controller value in the controlPlayer() function from false to true to play with a controller. The controls on the controller are the same as the keyboard, with left and right moving you left and right and up making you jump, just with the left analogue stick instead of arrow keys.



There are 2 known issues/bugs with this release of the game engine. Firstly, the collision system is not properly perfected, and therefore can cause some buggy things to happen. Here are some examples of the issues it causes:

* The player will not properly sit on the cloud platforms, and will instead hover a bit above them.
* Collision boxes are too big and can cause some annoying collisions.
* The spikey balls will sometimes be able to bounce on the player’s head without killing the player, and sometimes the player can jump off the spikey balls without dying.

The other bug with this release is a small visual one. When an object is facing left, the code is programmed to render the sprite flipped. This has caused some visual bugs for when the sprite is flipped, the most notable and consistent one being the bottom left pixel of the sprite not rendering properly. This is only a small visual bug and doesn’t impact the demo, but is a bug nonetheless.



**Maintenance Guide**

The game engine is split up into two sections: Visualisation, and Simulation.

*Visualisation*

Visualisation is used to add, store and manage the various sprite objects that are created to display in the window. It is responsible for telling all the stored sprites to render, and it figures out how they should render using the clipDetector structure, which is created for each sprite object that is created. All it does is store data for a box the same size as the sprite, so that visualisation can use it to detect when it needs to clip something.

The sprite object that is created through the visualisation class is responsible for actually rendering the sprite that is stored, as well as storing the sprite data and loading it from the given filepath. There are two ways to render a sprite: with alpha, and with no alpha. No alpha can be used for pure speed, but hasn’t been programmed to handle animation, unlike the function that renders it with an alpha.

The way the visualisation system is set up works very well for platformers, especially with the flipping of sprites, which the demo is able to take advantage of since it is a platformer. However, certain aspects of this system, such as the flipping of sprites based on their direction, doesn’t add to top down games, especially shooters.

*Simulation*

Simulation is used to handle everything to do with the game itself, including the physics, AI, entities and score. It is also used to add, store and manage objects, which are used to interact with the functions the simulation class has. It is also worth noting that simulation creates and stores a visualisation object, and has functions to call visualisation functions, but I will get onto that in a bit.

Simulation is linked to two structures and one class. Firstly, there is the object class, which is used to calculate movement and set the objects position so that it can used to set the position of sprites and the detectors. Since it is only used to store values, it could have been a structure instead, but I left it as a class so that functions can be programmed into it that the object has to calculate itself. Secondly, there is the collDetector structure, which is basically a box that is used to detect collision between solid objects. Finally, there is the entity structure, which is used to store health and damage for an object that has been declared an entity.

The game demo fully uses the various functions created to control and manage the various structures and objects created within the simulation. For example, it uses controlPlayer() to be able to move a specific object around as the player, and it also uses entityIsDead() to check if the player is dead, which it then uses to change game states. However, the functions created within simulation do lean heavily towards platformers, especially with the physics in the moveObject() function. However, it is structured well enough to make it relatively easy to program new functions into simulation to control the objects however you want.

*The identification system*

The entire game engine I have created revolves around an identification system, where every object, sprite, entity, collision box and clipping box has an identifier. It works much like a name; objects and sprites with the same name will be linked to allow for better management. For example, if you were to use the moveObject() function on an object with the name “Player”, and there was a sprite called “Player”, that sprite will be updated to the same position as the object.

This is done by running for loops through the vectors that store objects, sprites, etc. When it comes across an object/sprite with the same ID as the given ID, it will then run the appropriate code on that object/sprite. This is also why visualisation is linked to simulation, so that it can call the necessary functions with ease, and allows linking objects and sprites together to be much easier. It also can make the main code much neater.

The only disadvantage with this is that it can end up being slow, especially if there is a large number of objects and sprites stored within the vectors. This system might be able to be optimised by using while loops instead of for loops, but I have yet to find success with this, and therefore further research is needed.

*UML diagram*



**Conclusion**

In conclusion, I believe that the identification structure of the game engine is well implemented, allows for easy access to specific objects and sprites. It also allows other programmers to create and call functions for specific objects with little to no hassle, as well as create new functions that they can use to apply to specific object and sprites. Where I think this system falls short though is size; If there are too many objects/sprites, it can slow down very quickly, as it is looping through the entire vector due to it being a for loop. Another issue I can see happening is if two objects/sprites have the same name, as various functions will stop after they have found the first object/sprite with a matching ID, meaning that the second one will be completely skipped over.

There is plenty of room for improvement with the functions I have programmed into the engine to create the demo. The physics system is very buggy and can cause some dumb interactions to happen between objects. The rendering system doesn’t allow for animation on non-transparent sprites, and flipped sprites end up having some visual bugs. Furthermore, you have to call each individual object to apply a function to it, as it is not currently able to handle groups of objects apart from the score counter, which it creates itself. These are all issues I need to work on and improve in the future. I would also like to add more functionality such as rotating sprites, to allow for better support on other types of games like top-down shooters.

One of the main lessons I believe I learnt is to be wary of what you have to program in the future. I had to spend an entire week reprogramming the game engine just so I could complete milestone 5, as the previous system was inefficient and very limited in what it could do. Furthermore, there were numerous things I had to change in the milestones. For example, I had to change deleting platforms to respawning platforms, as I found that deleting from the heap memory while the game was running causes various issues.