**Group Report**

**Game Description**

We have named our game *Defiant Worlds* as there is resistance between two worlds; Earth and Mars. The genre of the game is a Real-Time Strategy Game with the overarching objective to destroy the enemy’s Command Centre.

As a player, you play as the faction *Earth Defence* and your mission is to protect your faction and Command Centre from the enemy, *The Crimson Legion*, and against an unknown faction; dubbed the *Rebels*.  
 Resources can be gathered by sending *Worker* units over to mine deposits, and with these resources the player is able to construct buildings, queue units, make repairs, and fight in space. The gameplay is split into two sections; ground warfare and space warfare. Ground warfare occurs when *Rebels* attack your base and attempt to cause disruption to your progress in an attempt to steal resources from you. *The Crimson Legion* can also send units over to your base (as can you send units to theirs) in order to try and thwart their defences and destroy their central command. Space warfare takes place when both factions have attempted to send over a fleet of ships which causes them to intercept one another. Units can be placed on transport ships and sent to space in order to reach Mars, however, if Mars has sent ships to Earth they can be intercepted. If a transport ship with units on dies, so do the units on board.

The player must weight up their strategy over having enough ground units to defend against Rebels or losses in space, whilst maintaining a large enough fleet to send troops to Mars as well as defend from Mars attacking and sending in their own troops.

**Technical Details**

The game is split into 3 game states; Main Menu, World, and Space. The game’s state is updated based on the user’s requests to save, load, send a fleet into space, or even starting a new game.

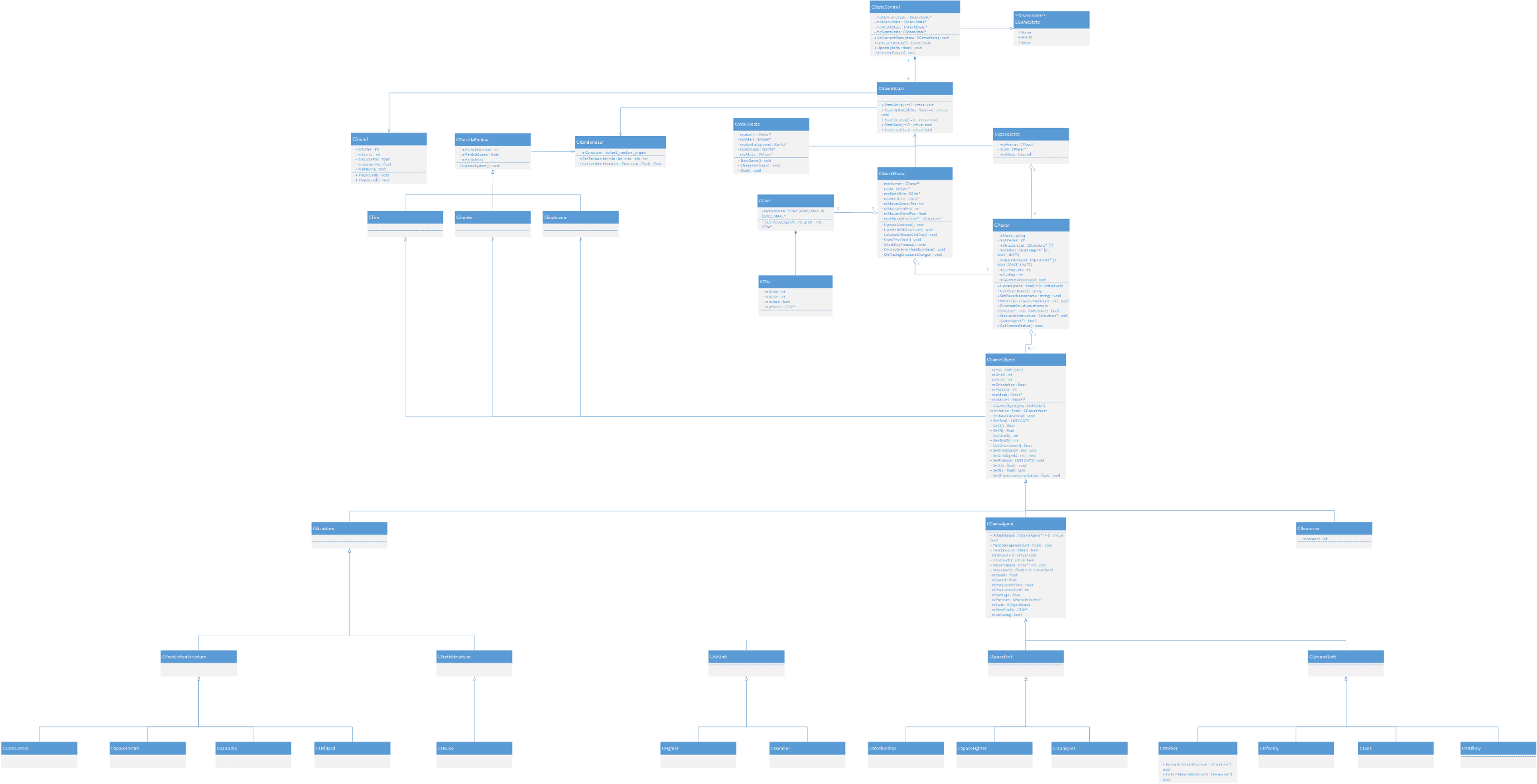
In order to implement the structures and units, polymorphism was used. Base structures and units share a common ancestor; CGameObject containing position, health, and model details. Each type of structure inherits from the CStructure class, split into 3 child classes; CProduction, CStatic, and CDefense. Structures such as the Barracks, Hellipad and Space Centre are instances of CProduction, which stores each structure’s queue of units, what units they can build, their Mesh and various other members. Wall structures inherit from CStatic as they do not perform any actions. Anti-Air turrets positioned on the corners of the walls on pillars inherit from CDefense as their primary function is to shoot down aircraft that fly over the walls.

Similar to how structures were split into child classes, units had the same. All units inherit from the base class CGameAgent. There are 3 types of units; CAir, CSpace and CGround. In the air units there is a CBomber class and a CFighter class each with their own Update function determining how they are able to attack. Bombers can only attack ground units and have a slow fire rate but high damage output. Fighters on the other hand have a very fast fire rate, low damage output and can attack both ground and air targets. Units inheriting from CGround have similar attributes to that of air units.

In order to build the structures, the player has to click a button (or press a key from 1 to 3) to select the structure they want to build, and click on the grey area where they want to place it. Ray casting has been implemented in order to determine where in the world the user is attempting to click when they try and place a building or select a unit. Furthermore, a grid has been implemented to ensure that buildings do not overlap (or placed in the same position) when they are placed.

The final overview of the technical implementation is the implementation of the player class; each player has a hash map of their units and structures for efficient access and storage. The Ai Player class is a child of the standard player class which has its own Update function, used to determine what decisions the AI should make and carry out tasks which the AI has queued for itself using a priority queue.

**On the next page is the class diagram for our project. I higher resolution one can be found in the ZIP folder.**

****