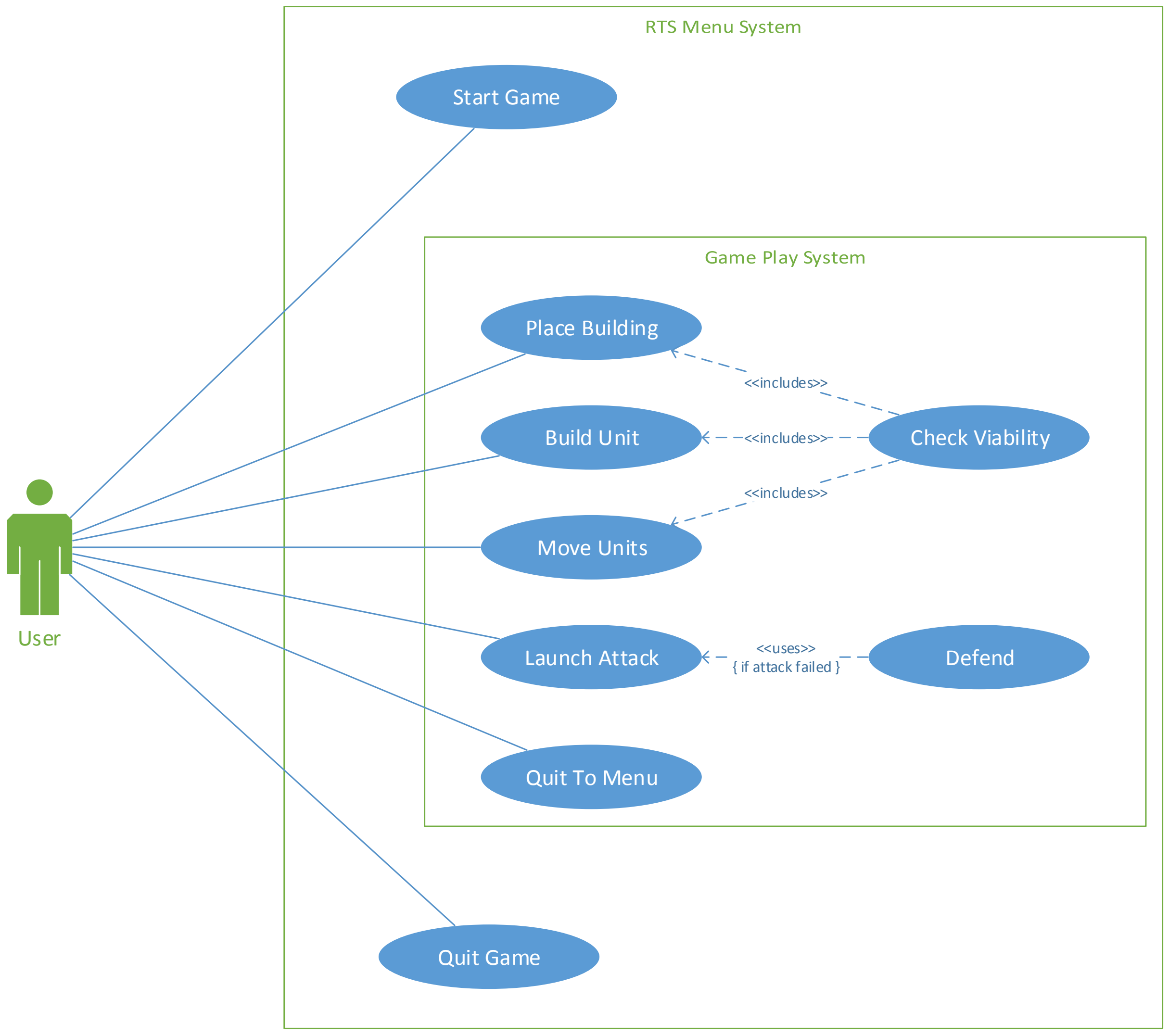
Part 1: Design Documentation

Use Case Diagram



Sam Hignett (G20599224) Michael Hinchcliffe (G20537816) Kyle Hobdey (G20594424)

Brief Textual Overviews

Use Case Name:
Start Game

Actors:
User

Use Case Description:
After the user has launched the game, they are greeted by the main menu. The main menu itself has buttons representing the loading of the game, starting a new game, editing game options, or quitting the game entirely.
Upon clicking the ‘Start New Game’ button, the base grid for both the Earth are and Mars area is loaded. In the centre of the each grid is the respective faction’s Command Centre, and randomly dotted around the map are resources which can be gathered. The User Interface is loaded at this point, as well as the initialising of the AI.

Use Case Name:
Place Building

Actors:
User

Use Case Description:
The game is in play, and the user has selected a building from the User Interface which they want to construct. The user will click on a tile that they want the building to be placed. Upon doing this, various methods are called to ensure that the location the user wants to place this building is valid. If it is valid, the building will be placed. Otherwise, a message will be shown on the bottom left of the screen in red font.

Use Case Name:
Build Units

Actors:
User

Use Case Description:

Whilst the game in is play, the user is able to utilise the buildings they have created in order to construct units which will be used to launch attacks on the AI player.
When the play clicks on a constructed building, they will be shown a list of units that can be built from that particular building. From this, if the building is free for production, they will be able to either click the image of the unit or press a hotkey designated to that unit to build it. When a unit is selected, the building will be in production for the duration of the unit’s production time.

Use Case Name:
Move Units

Actors:
User

Use Case Description:
Throughout the game, the player will have the ability to manoeuvre his/her units around both earth and if there are able to launch a successful invasion, Mars. The player will have different unit types to choose from, and as such will have multiple movement types. Throughout all the types of units though is the player's ability to move them to where they are needed. The player will be able to click on units to select them, and right-clicking on a position on the map with a unit selected will instruct the unit to move to that position. If the position is being used by a building or another unit, the moving unit will attempt to get as close as possible to that position.

Use Case Name:
Check Viability

Actors:
User

Use Case Description:
The game is in play, and the user has attempted to build or place something somewhere. The game then ensures that the user has the appropriate resources to complete the task, and returns a fail message if they do not. It then checks to see if the there is enough space to place a building, and returns a fail message if there isn't, and allows the user to reposition the building or stop attempting to place a building.

Use Case Name:
Launch Attack

Actors:
User

Use Case Description:
Whilst the game is in the world state, when the user decides to attack, they are given the option to select a number of ships to attack with, or cancel the attack. If they choose to cancel then they are returned to the world state, else the game moves to the space state and loads the number of ships the user selected. These then fight with the enemy ships, which were loaded in at the same time. After the battle resolves it returns to the appropriate world state.

Use Case Name:
Defend

Actors:
User

Use Case Description:
Whilst the game is in play, the user will be at constant battle from the AI player. If the player loses a space battle, then the AI player will land units on earth and will try to destroy the player's command centre in order to win the game. The player will have to build and command units in order to try and defend his/her base from destruction.

Use Case Name:
Quit To Menu

Actors:
User

Use Case Description:
Upon the game being paused, the user is faced with various options; 'Resume', 'Save' or 'Quit'. When 'Quit' is selected, the user is taken back to the main menu. Everything in the main menu state is created and displayed, whilst those in the playing state are unloaded and destroyed.

Use Case Name:
Quit Game

Actors:
User

Use Case Description:
Whilst in the main menu, if the user presses escape or clicks the quit button, they are pre presented with the option to either cancel or quit. If cancel is selected, they return to the main menu and nothing happens, however if quit is selected then the game unloads everything and the application closes down.

Detailed Textual Descriptions

Use Case Name:
Place Building

Actors:
User

Triggers:
User clicks whilst holding a building to place.

Preconditions:
User is holding an object of type structure, and cursor is within the game grid.

Post-conditions:
The surrounding area is checked to ensure it is a valid location.
Player's mineral amount is checked to ensure enough is owned to pay for building.
If the location is valid and enough minerals are owned, then the building is placed.
Flags are raised in the grid locations the building encompasses.
Cost of building deducted from player.

Successful Flow:
1. User selects a building to place
2. The user hovers mouse over grid square in which they want the building to be placed.
3. The game compares the cost of the building to how many minerals the player currently has.
4. The game accepts that the player has sufficient funds.
5. The game checks the surrounding area which the building requires in order for the structure to be built.
6. The game passes the check – surrounding area is clear of other structures and is not overlapping the edge of the game grid.
7. The game creates the building at the chosen location.
8. The game deducts the cost of the building from the player's funds.
9. The user continues to play.

Unsuccessful Flow:
2A1: The player does not have enough minerals to pay for the cost of constructing the building. An error message is returned at the bottom left of the screen letting the player be aware of the issue.

4A1: The area surrounding the building trying to be placed is being used by other structures, making it an invalid placement. An error message is returned to the player, displayed at the bottom left hand side of the screen.

4A2: The building overlaps the edge of the map, thus an invalid building placement. Again, an error would be displayed at the bottom left of the screen highlighting the issue.

Use Case Name:
Build Units

Actors:
User

Triggers:
User either clicks on unit image or presses hotkey with building selected to build unit.

Preconditions:
User has selected a building which can produce a unit

Post-conditions:
The building's current production is checked to determine if it is already in production
Player's mineral amount is checked to ensure that the unit can be bought.
If the location is valid and enough minerals are owned, then the building begins production.
Cost of unit deducted from player.

Successful Flow:
1. The user selects the building they want to produce from.
2. The user selects the unit they wish to produce.
3. The game compares the cost of the unit to how many minerals the player currently has.
4. The game accepts that the player has sufficient funds.
5. The game checks the building is not already in production of a unit
6. The game passes the check – building is currently not producing a unit
7. The building begins production of the unit.
8. The game deducts the cost of the building from the player's funds.
9. The user continues to play.

Unsuccessful Flow:
3A1: The player does not have enough minerals to pay for the cost of producing the unit. An error message is returned at the bottom left of the screen letting the player be aware of the issue.

5A1: The building is already producing a unit, and as such cannot produce two units at the same time. The unit is placed in the production building's build queue. When the production building's queue reaches the unit, it will start production.

Use Case Name:
Launch Attack

Actors:
User

Triggers:
User clicks attack button

Preconditions:
User is in the earth state

Post-conditions:
The player units are checked to ensure they have ships
If the player has ships, then the game moves to the space state
The player and enemy ships are loaded in
The space state enters the attack stage

Successful Flow:
1. The user clicks the launch attack button
2. The game checks to see if the player has any ships
3. The game asks how many ships they want to commit, or if they want to cancel
4. The game Loads in the ships
5. The game enters the attack stage

Unsuccessful Flow:
2A1: The player does not have any ships with which to launch an attack

3A1: The Player chooses to cancel the attack, and the game returns to its previous state

Noun-Phrase Analysis

Defiant Worlds

Description:
The aim of the game is to destroy the enemy's Command Centre. You do this by placing buildings and using them to produce units. These units can then attack other units of similar type, and can also be used to defend from attacks.
Each player must collect resources, in the form of minerals, in order to construct buildings and produce units. The resources spawn randomly throughout each player's planet. The only way the minerals can be gathered is through using a Worker unit. In order to attack the enemy player, units must be sent through space where they can be intercepted by the defending player.
During space battles, players select tactics which can help determine the outcome of the battle. If you are attacking and you win the space battle, or there is no interception, your units land on enemy's planet, and can attack their buildings.
Each unit has its own build time, damage, cost, health and type. Same applies to the buildings, apart from damage.

Candidate Classes

- Destroy
- Command Centre
- Placing
- Buildings
- Produce
- Units
- Attack
- Player
- Type
- Defend
- Collect
- Resources
- Minerals
- Spawn
- Planet
- Enemy
- Space
- Intercept
- Battles
- Tactics
- Win
- Build time
- Damage
- Cost
- Health

Domain Classes

- Command Centre
- Building
- Units
- Player
- Minerals
- Planet
- Enemy
- Space

Command Centre

The Command Centre is the main building for each player. From this building the player and the AI will be able to construct Workers. The Command Centre will be the key target for the other player, as destructions of the Command Centre will end the game. Each game will start with the Command Centre for each player in the centre of the map.

Building

As with most RTS games, Buildings are a key aspect of gameplay. From buildings, the player will be able to construct units which will be used to launch attacks upon the AI player. There will be several types of buildings which will have their own set of units which that particular building type can produce. There will also be buildings in the background of each world for aesthetic purposes.

Units

Units are the core component of RTS games, and are essential for both players to achieve victory. Units will be constructed from buildings and used to wage war on the other player. The units will be able to be moved around each world and ordered to attack the other player's units and buildings.

Player

There will be two players within the game. The human player and the AI player. Both will have the ability to construct buildings and units, as well as launching attacks upon the other player and command units to move and attack.

Minerals

Minerals will be the main resource for both players. Minerals will be placed in both worlds and will be mine-able. Workers will be used to harvest them, and with Minerals players will be able to purchase and construct buildings and Units.

Planet

There will be two planets which will be home to the two players. Earth will be the home planet for the human player and will be the area in which the human player will building his base and army. Mars will be the same but for the AI player, and players will be able to go to the other player's planet after a successful space battle.

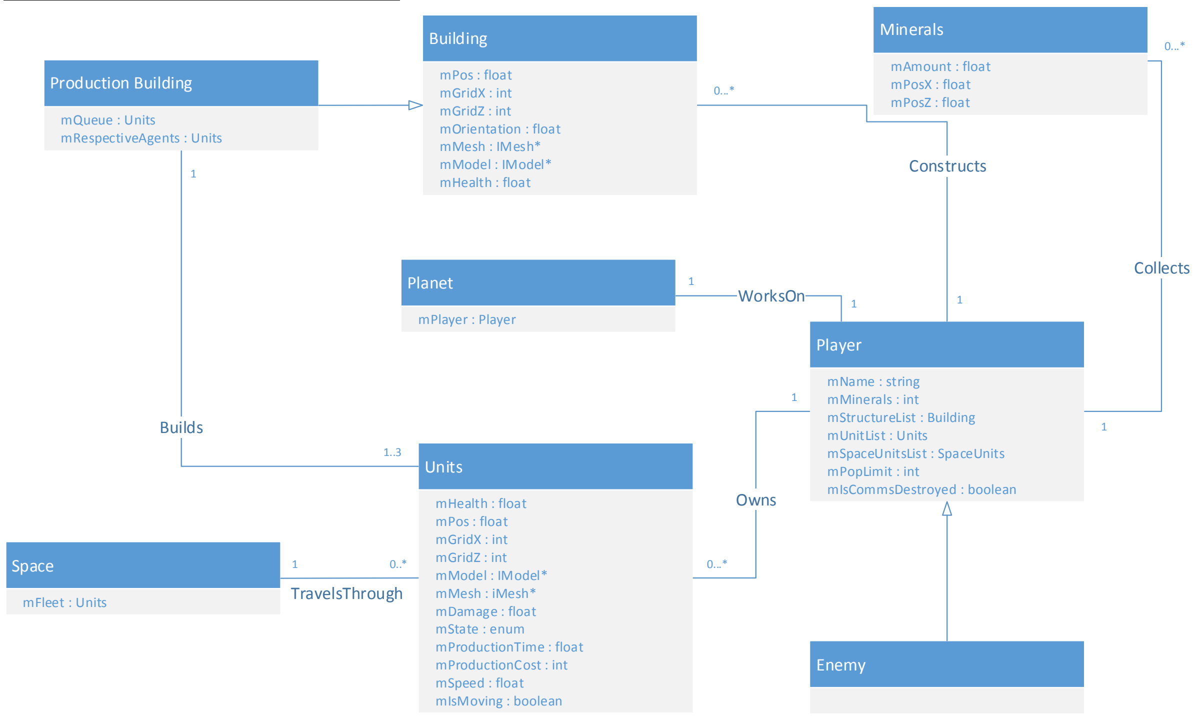
Enemy

Enemy, in this context, will mean the other player. Both players will be able to attack the other player's buildings and units.

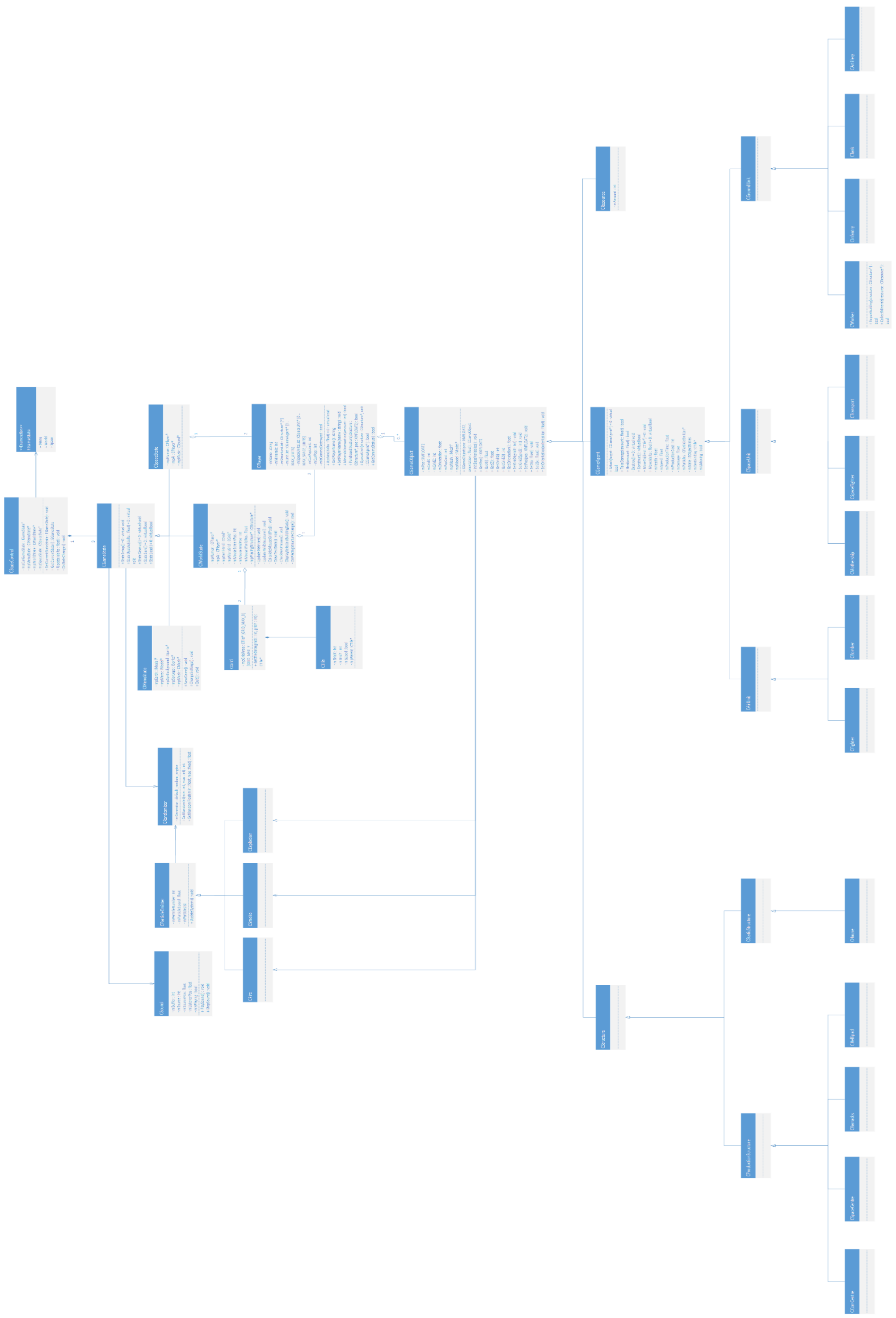
Space

Space is a state within a game in which both players will fight their space units against each other. The players will be able to choose the tactics in how they will engage the battle and then the victor will be able to launch an invasion onto the enemy's planet with the units they have remaining.

Domain Class Diagram



Sam Hignett (G20599224) Michael Hinchcliffe (G20537816) Kyle Hobdey (G20594424)



CRC Cards

CGameObject	
Subclasses: CGameAgent, CMineral, CStructure	
Responsibilities	**Collaborations**
Movement	CGrid
Handling game object's grid & world position	CSound
Implement model's scale	
Track model's orientation	
Store which faction the game object belongs to	
Know which sound the object makes & when to play it	

CGameAgent	
Subclasses: CAirUnit, CGroundUnit, CSpaceUnit **Superclass: CGameObject**	
Responsibilities	**Collaborations**
Monitor state of the game agent	CParticleEmitter
Update state of the agent	CResource
Perform AI calculations when moving	
Turns on particle emitter when firing shots	

CResource	
Superclass: CGameObject	
Responsibilities	**Collaborations**
Update amount of resources a player has when being collected	CPlayer

CStructure	
Subclasses: CProduction, CStatic **Superclass: CGameObject**	
Responsibilities	**Collaborations**
Update max number of units a player can have	CGameAgent
Know which units can be built from a given structure	CParticleEmitter
Monitor state of the structure	
Update state of the structure	
Emit smoke particles when low on health	

CParticleEmitter	
Subclasses: CSmoke, CExplosion	
Responsibilities	**Collaborations**
Handle creation of particles	
Update particle movement	

Track each particle's lifespan	

CSound

Responsibilities	Collaborations
Load sound files	
Play sound files	
Know which sounds are to be looped	

CStateControl

Responsibilities	Collaborations
Monitor state of the game	CGameState
Update current state	CPlayerManager
Handle the closure of the game	
Manage player states	

CGameState

Subclasses: CMenuState, CWorldState, CSpaceState

Responsibilities	Collaborations
Update events in the respective state	CPlayer
Handle UI	CSound
Play background music	CButton
Determine win/loss	CGameObject
Hold grid data	
Contains relative game objects	

CPlayerManager

Responsibilities	Collaborations
Know how many players there are	CPlayer
Know when players have been created	
Run decision algorithms for AI players	

CPlayer

Responsibilities	Collaborations
Store owned structures	CStructure
Monitor finances	CGameAgent
Store owned units	CFleet

CFleet

Responsibilities	Collaborations
Store space units for a given player	CSpaceUnit
Monitor status of each unit	CPlayer

CRandomiser

Responsibilities	Collaborations
Generate a random number	

CGrid

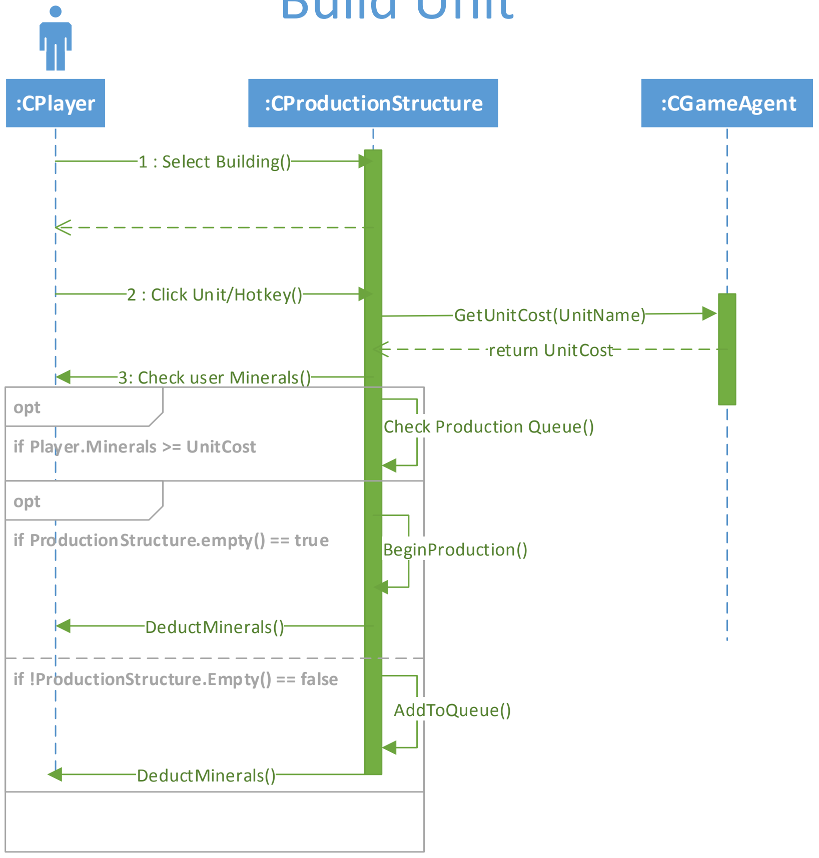
Responsibilities	Collaborations
Know grid dimensions	CTile
Store instance of CTile per grid square	
Be able to retrieve data for a given tile square	

CTile

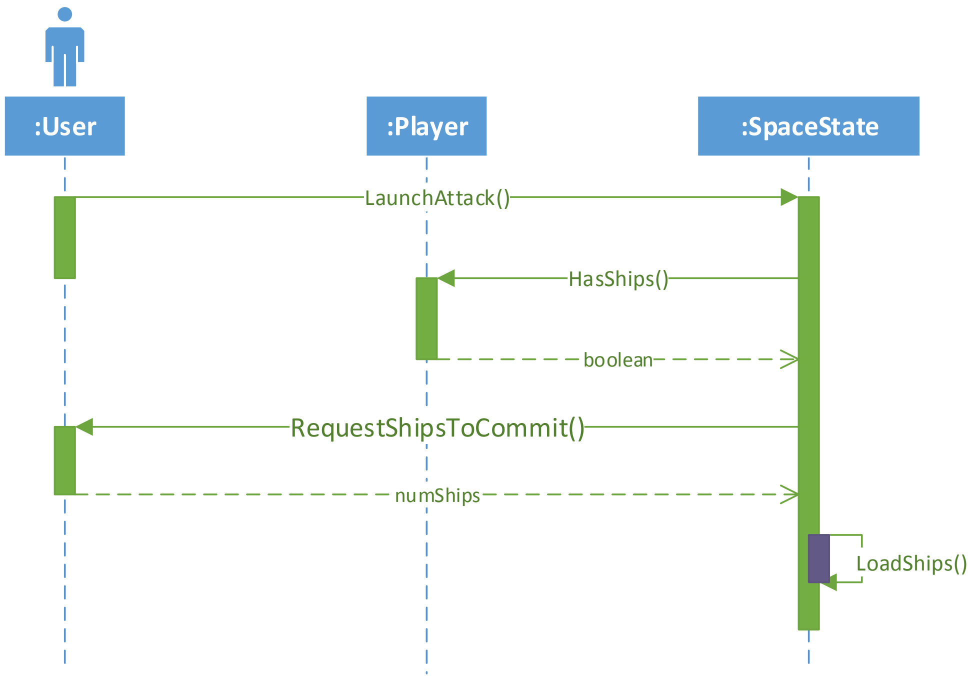
Responsibilities	Collaborations
Track tile status	
Update tile status	
Know position in world and in grid where this tile falls	

Sequence Diagrams

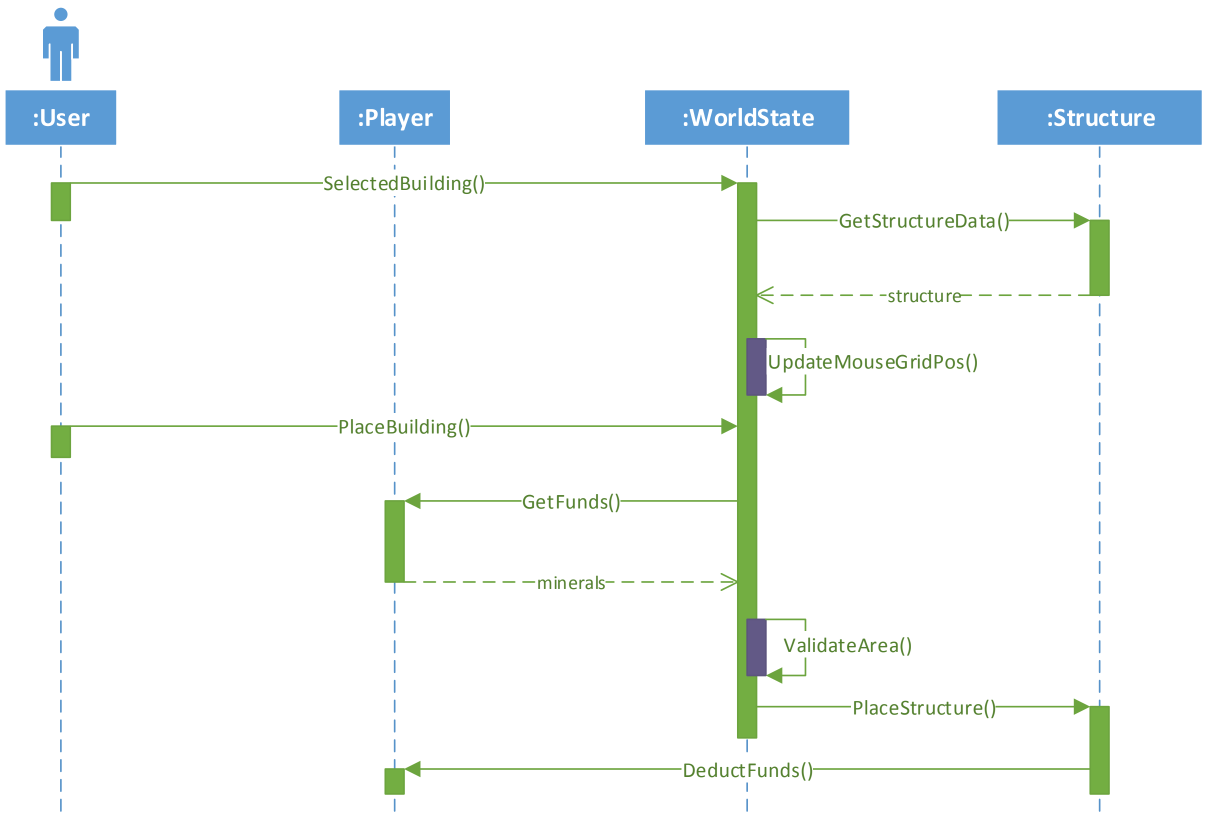
Build Unit



Launch Attack

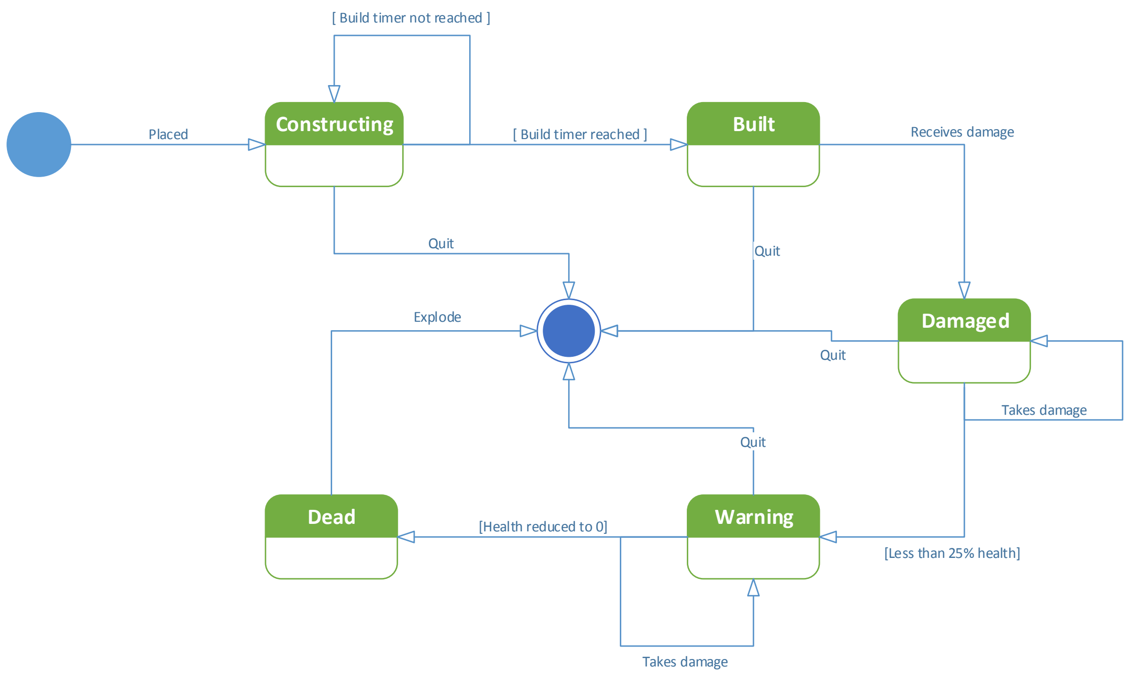


Place Building

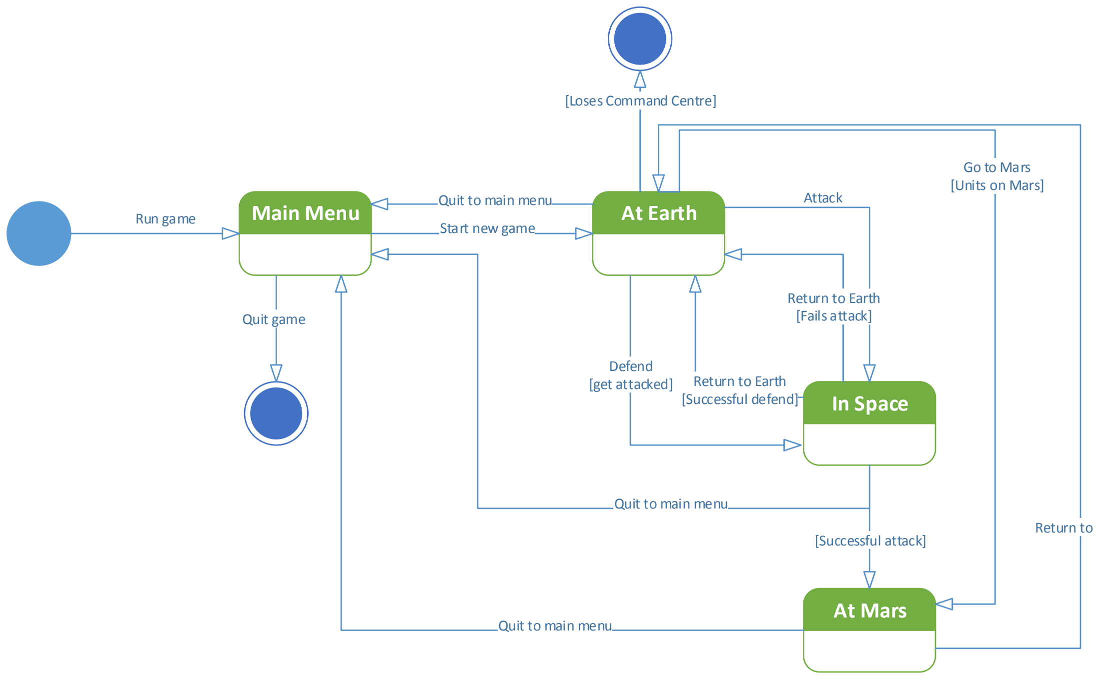


State Transition Diagrams

CStructure State Transition Diagram



CPlayer State Transition Diagram

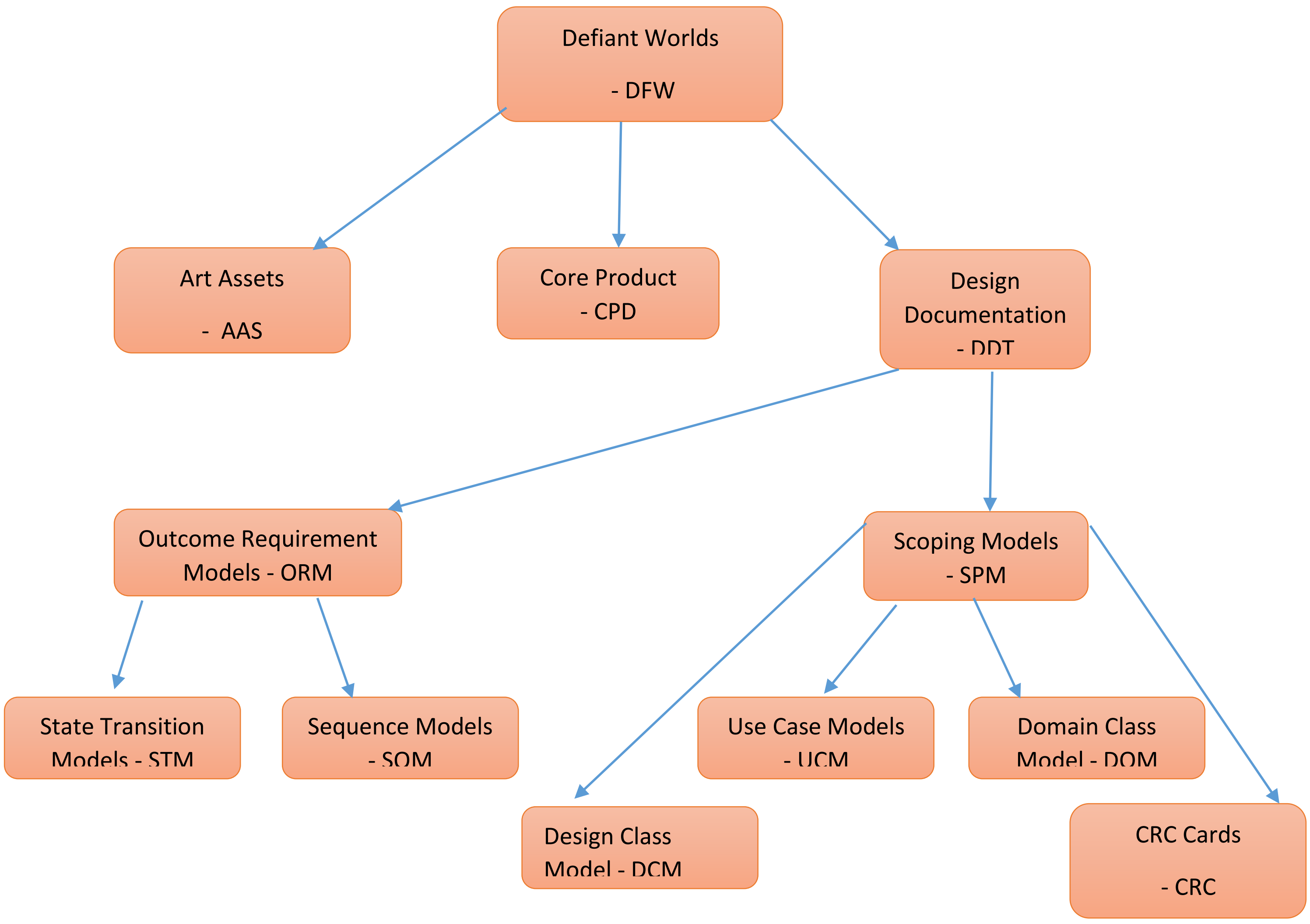


Configuration Management Plan

Identification Scheme

Through the project, there will be several different aspects which will need to be managed. As well as the actual product development, all design documentation, including diagrams and models, will need to be managed and controlled. Also all assets required for development will have to be managed.

ID Scheme



Responsibilities

As there is limited time for project completion it has been assigned such that each developer will have a responsibility to manage the changes and the versions of the various manageable items. Using the version control and configuration management (CM) tools we have decided to use, developers will be able to control the CM procedures.

Also communication will be key and as such all developers will be contactable should confusion around version control arise.

During routine team meetings, the entities submitted to the group central repository will be discussed and managed, as well as configuration assurance audited as part of a team.

Version Management Policies

For Version control and management, the group will work on pre-determined areas of the project that they have been assigned to. This ensures that the overlap involved will be minimal and so will cut down on merging errors. Any new errors, with priority, will be mentioned in the title of the most recent version of the outcome, and all known issues will be mentioned and described within the main.cpp file.

After a set amount of features are released or a significant number of issues are corrected, a new baseline will be constructed. From this we will be able to re-assess the requirements of the new baseline and to prioritise features or issue fixes.

After the initial development phase, we will then use CM for defect management and run the solution through pre-determined tests. Upon a test failure, a note will be made within the repository and then necessary changes can be made.

Version Management Tools

The main version control tool we will use is Git, which is a web based repository hosting service. Through this we will be able to manage the source code for the project, as well as have a hosting repository for the rest of the manageable files and perform all required CM procedures.

The Git repository will be used by all team members. When starting development sessions, the procedure will be that ‘fetches’ and ‘pulls’ will be done by all developers, to ensure that there is the lowest chance of merging issues when it comes to ‘pushing’ the changes to the central repository.

Throughout the development session, the procedure will also be to contact the other developers, should a sudden issue arise. This will be done to ensure that if the central repository is compromised, that the local copies from developers can be used as a restore point.

At the end of each development session, a final fetch and pull should be done to ensure that conflicting edits do not occur or are resolved before committing changes.

Configuration Database

The configuration Management database will be the central location in which faults and changes will be made. Within our group, all current faults will be documented within the main page of the solution so that developers can be made aware of issues when starting a development session. One of the main benefit Git, the web based repository, is that it tracks all changes between repository versions. This will be what is used to track repository changes.