Civilization Wars: Advanced

Final Project Report

B4

Alex Bor

Mattie432

Luke Richardson

Dhruvil Tank

Robert Zlatarski

[Introduction](#h.hxdf7hjp8yva)

[Software Design](#h.lw8v60lcbjd1)

[Game model](#h.4vjymswhkew4)

[Class description](#h.gmp4tzfhhow1)

[Game Design](#h.be3w5278jeof)

[HCI (Human-Computer Interaction)](#h.ltvplmr4ey2m)

[GUI (Graphical User Interface)](#h.ey2kp61175nj)

[AI (Artificial Intelligence)](#h.np2b7mvssysf)

[Networking](#h.ihj601ptvv1r)

[Software Engineering](#h.uvfravb6f54d)

[Class Diagram](#h.k5mif4cbwf7v)

[User Interface Design](#h.pxrn6q49weeg)

[Project Life Cycle](#h.cviw2vm11ibq)

[Risk Management](#h.ws4nk261rxlx)

[Evaluation](#h.syr6c7wyjwff)

[Teamwork](#h.upvr06aonwk3)

[Organization](#h.gzkvc1o0hkba)

[Problems](#h.txfd9rhwx9i9)

[Teamwork system](#h.fdlv251hisko)

[Gantt-charts](#h.ldit745r2wb1)

[SVN activity](#h.guxwhldkckv7)

[Summary](#h.v20g52dhbba8)

[Individual Reflection](#h.125a4u6nqdvi)

[Alex Bor](#h.izr9tkypfv)

[Mattie432](#h.sbgsb36blm3)

[Luke Richardson](#h.az4q1r1qyjxv)

[Dhruvil Tank](#h.lyw29p7cefox)

[Robert Zlatarski](#h.qziyqiyqdvj4)

[Appendix](#h.xqy19gyymvyo)

[Class diagram](#h.19rhoitbcihs)

[Gantt Chart](#h.kwbydv7wabsl)

# 

# Introduction

Civilization Wars: Advanced is a turn based strategy game. It is an intuitive and easy to play game that is suitable for everyone. As a strategy game, it requires the player to think logically in order to beat his opponents. Once you start the game, you get the opportunity to choose to play either versus an AI opponent or versus players on your local area network.

The game basically requires players to build a base where they build their buildings and units and eventually defeat all of the other players that are not in their team. That is a really good and fun way for people to develop their logical thinking, because the game requires from you to start thinking a few moves ahead (like a chess game). As it is a turn based game, each player has a number of things he can do during his turn, which include: create new buildings, spawn units, move units and more. It is important to be noted that in our game, when you create a new tank for example, that is not only one tank, but it represents a squad of tanks. Movement of units is limited to their range and different types of units have different ranges.

There are also different cell terrains, each of which has buffs for units that are onto it. It is very easy to go around the map and there is also a mini map that is a real time representation of the map, which helps you to have a better view of the whole map at any time, but scaled to be fit in the bottom context bar. At the end of your turn, you receive a fixed amount of money and a new “game day” begins.

The game is really easy and fun to be played and if a player has difficulty in understanding the game rules, there is an instructions menu that will bring you the instructions screen, where everything that you are required to do as a player is described.

The final project report includes:

* **Software Design** - outlines the purpose of the classes and the methods and constructors it encases.
* **Game Design** - outlines the game map, with all of its different terrains, and also a description of the units and buildings.
* **Human-Computer Interaction** - outlines what different buttons of the keyboard do and how each in game button reflects on the gameplay and the different buttons a player can see when he selects different units and buildings, given an example from another strategy game.
* **Graphical User Interface** - describes the game’s top bar and bottom menu and their functionalities.
* **Artificial Intelligence** - describes how and AI player moves in a given game. There are a few examples of AI game styles and also an outline of the AI algorithm.
* **Networking** - explains how the networking in the game works and what are its benefits.
* **Software Engineering** - includes the game’s class diagram and its description, the initial user interface design and the project life cycle.
* **Risk Management** - outlines what were the project priorities and how collaboration between team members has been done.
* **Evaluation** - final thoughts over the game our team has produced.
* **Teamwork** - outlines team organization, problems that the team has encountered, how we used our initial Gantt-chart and a summary of the SVN activity.
* **Summary** - includes the team’s final thoughts over the game development.
* **Individual Reflection** - carried out by Alex Bor, Mattie432, Luke Richardson, Dhruvil Tank and Robert Zlatarski.

# Software Design

## Game model

Our class structure was based around the idea that there was one class that should be ‘master’ and manage the others. For the starting menus this is ‘MainMenu’, here all of the operations of the main menu are performed such as choosing a game type and viewing the instructions. The main menu also launches the game however the game itself is controlled through the ‘Game’ class. The ‘Game’ class collects all of the components of a game together (e.g. UI, players, map) and acts as the central manager for them.

Each class is clearly defined to fulfil its own role, for example the ‘Building’ class manages everything to do with buildings. This helps to keep the project ordered and makes it easier to code and understand the code as there is not unrelated methods inside a class. We initially defined this structure in the specification and it has worked well throughout the project.

There are also many external resources that the game needs to function such as images for the map, units and buildings and sounds for the menu and game music. All of these have been imported into the project and are packaged with the game when its exported as a jar. They are organised into folders of images and sounds and then sub-folders for better referencing.

## Class description

Below there is a table containing all of the classes in the project as well as their type, a description of the class and and sub/super classes associated with it. The project is hierarchical in nature as some classes are derived from others. This was apparent from the class diagram that we produced in the preliminary documentation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Type** | **Sub**  **classes** | **Super class** | **Description** |
| Building | Abstract | Base  Factory  Barracks |  | This is an abstraction for the different types of buildings in the game. As they all share some common features this class contains information that applies to all of its subclasses. Examples of this may be ‘occupying cell’, ‘owner’, ‘health level’ as well as methods such as ‘placeBuilding’. |
| Base | Concrete |  | Building | This contains methods for actions specific to the base, such as ‘buildFactory’ and ‘buildBarracks’. Its constructor also sends the relevant parameters to its superclasses constructor. |
| Factory | Concrete |  | Building | This contains methods for actions specific to the factory, such as ‘recruitTank’. Its constructor also sends the relevant parameters to its superclasses constructor. |
| Barracks | Concrete |  | Building | This contains methods for actions specific to the factory, such as ‘recruitMarrine’. Its constructor also sends the relevant parameters to its superclasses constructor. |
| FocusTraversalOnArray | Concrete |  |  | This is a library class created by the window builder plugin used to aid GUI creation. This is available under public license. |
| Game | Concrete |  |  | This contains many of the core methods fundamental to the operation of the game. This ranges from organising which player is currently playing to ending the day when time runs out. |
| GameType | Enum |  |  | This is an enumerator class for the different types of game type that we have implemented. It also holds the difficulty level when playing versus AI. |
| Teams | Enum |  |  | This enumeration contains information regarding the teams of the players in the game. It tracks which players are enemies and allies, the number of players in the game and whether they are human opponents or AI controlled. |
| Map | Concrete |  |  | This configures and controls the map, it is used to set up the map when the game is started and hold all information relating to it (examples include ‘array of map cells’, ‘number of cells across’ and the ‘mapName’. It also contains methods such as ‘getPlayerUnits’ and ‘removePlayersBuildings’ and others which apply to the map as a whole. |
| Cell | Concrete |  |  | This class represents one grid cell on the map. It holds all information about that cell such as any units or buildings it contains and methods such as ‘draw’ and ‘getSurroundingCells’. |
| Terrain | Abstract | Bridge  Flat  Forest  Mountain  Water |  | This is an abstraction for the different types of terrain in the game. It contains methods and variables common to all such as ‘attackBuff’ and ‘getAllowedBuildings’ and is used in cell as each cell can have a different type of terrain. |
| Bridge | Concrete |  | Terrain | This class contains a constructor that sets the variables in the superclass to what they should be for this terrain type. |
| Flat | Concrete |  | Terrain | This class contains a constructor that sets the variables in the superclass to what they should be for this terrain type. |
| Forest | Concrete |  | Terrain | This class contains a constructor that sets the variables in the superclass to what they should be for this terrain type. |
| Mountain | Concrete |  | Terrain | This class contains a constructor that sets the variables in the superclass to what they should be for this terrain type. |
| Water | Concrete |  | Terrain | This class contains a constructor that sets the variables in the superclass to what they should be for this terrain type. |
| AcceptConnections | Concrete |  |  | This class listens for connections when the host starts a game. It runs in its own thread and is continually running until a game is started or quit. |
| Client | Concrete |  |  | This handles all of the main tasks for a client. Examples include ‘sendMessage’ and ‘sendMoveTo’. |
| ClientEndTurn | Concrete |  |  | This class sends an ‘endTurnPacket’ to the server. It is used to signify that the client has either clicked end-turn or that there time has run out and the game should move to the next player. |
| ClientGamePacket | Concrete |  |  | This is a packet object for sending the game information to all connected clients. It sends the map choice as well as the player order and game type. |
| ClientSend | Concrete |  |  | This sends moves to the server to then be distributed to all connected clients. |
| ClientSendMessage | Concrete |  |  | This sends text messages to the server to then be distributed to all connected clients. |
| DataSent | Concrete |  |  | This class is a packet object for the moves of the player to be sent to the host. |
| EndTurnPacket | Concrete |  |  | This class is a packet object for the end turn command to be sent to the server. |
| MessagePacket | Concrete |  |  | This class is a packet object for a message sent command to be sent to the server. |
| Server | Concrete |  |  | This is the host of the networking protocols. It receives messages from clients and distributes them to all clients. It handles things like the ‘staring of multiplayer games’ and ‘player moves’ sent from within a game. |
| ServerListen | Concrete |  |  | This is the listener for the server it is continuously running and listens on a socket for messages from clients. |
| Player | Abstract | Human  AI |  | This is an abstraction for the different types of player in the game. This class contains common methods to all, examples include ‘getMoney’, ‘getPlayerName’ and ‘canAfford’. |
| Human | Concrete |  | Player | This class contains a constructor that sets the variables in the superclass to what they should be for this player type. |
| AI | Concrete |  | Player | This class contains a constructor that sets the variables in the superclass to what they should be for this player type. It also contains methods for the AI processing, such as ‘createBuildings’, ‘moveUnits’ and ‘play’. |
| Sound | Concrete |  |  | This handles the playing of all music in the game. It is used to play any sounds and can be changed at any time. |
| Unit | Abstract | Marrine  Tank |  | This is an abstraction for the different units in the game it contains methods applicable to all units such as ‘attack’ and ‘move’. |
| Marrine | Concrete |  | Unit | This class contains a constructor that sets the variables in the superclass to what they should be for this unit type. |
| Tank | Concrete |  | Unit | This class contains a constructor that sets the variables in the superclass to what they should be for this unit type. |
| MapPanel | Concrete |  |  | This extends JPanel and is used to display the map to the user. It performs all of the drawing of the map, units and buildings as well as storing information about where the user is looking on the screen. |
| MapPanel\_MouseListener | Concrete |  |  | This extends MouseAdapter and contains all of the methods for various mouse click events on the map panel. Examples include ‘clicking a unit’ and ‘clicking a building’. |
| MapPanel\_KeyListener | Concrete |  |  | This contains all of the methods for the KeyPress events associated with the MapPanel. Examples include ‘scrolling with the arrow keys’ and ‘zooming with + and -’. |
| UserInterface\_Bottom | Concrete |  |  | This contains methods associated with the interaction with the bottom menu bar. Examples include ‘endGame button clicked’ and ‘recruitTank button clicked’. It also contains methods to update various components when a user clicks on a map cell, for example ‘setTerrainBar’ shows the stats for the terrain when an empty cell is clicked. |
| UserInterface\_Top | Concrete |  |  | This handles the top bar of the user interface. It controls what is seen and displays a countdown of remaining time for that player. |
| Instructions | Concrete |  |  | This creates a new JFrame and displays all of the instructions for playing the game to the user. |
| MainMenu | Concrete |  |  | This contains the main method for the game. It controls what is shown to the user in the main menu, for example ‘singleplayer’, ‘multiplayer’. Also starts the game based on what the user has chosen as the game parameters (e.g. ‘map’, ‘difficulty’). |

# Game Design

Our game functions like most strategy games in respect to we have a map that buildings will spawn on and then we can create units from the buildings and finally we can control the units to move and attack. In this section I will be outlining the different terrains building and units that we use and how they interact with each other.

Our map was designed using five different terrains that have different properties when we are spawning units and buildings or moving around the map with our units. The terrains we have include:

* Flat - Our basic land that takes up the majority of the map. Flat allows any unit or any building to be placed on it or move across it.



* Water - Only tanks can pass through this terrain.



* Bridge - A way for units to cross the Water terrain. Only marines can move across this terrain.



* Forest - Only marines can move across this terrain.



* Mountains - Only marines can move across this terrain.



In order for the player to be able to play the game and win we created three types of buildings and two types of units that the player can interact with. The player can create new buildings and units by interacting with the current buildings on the map, the player can also interact with the units by either moving the units around the map or attacking other units or buildings.

Buildings:

* Base - The main building, each player starts with one Base and they cannot create another Base and when the Base is destroyed that player loses. The player can create the Factory or Barracks within range of the Base using the GUI buttons that handle the Base’s functionality.



* Barracks - This building is used to create Marines. Only four Marines can be spawned around the Barracks giving preference to the x-axis then the y-axis.



* Factory - This building is used to create Tanks. Only four Tanks can be spawned around the Factory giving preference to the x-axis then the y-axis.



Units:

* Marines - The basic unit, the Marines have no limit on how many can be created (while the player has the money to create them). As mentioned before Marines cannot cross water however they can cross all other forms of terrain.



* Tanks - The power unit of the game, the player has a maximum limit of five Tanks on the map at any one time.



The buildings and units have stats that determine their effect on the game. For buildings they have stats for Health, Cost and Reward. For units they have stats for Health, Cost, Reward, Damage, Number of Units and Range.

The health stat track how much damage a building/unit can take before it is removed from the game.

The cost stat tracks how much money the player needs to create that building/unit.

The reward stat tracks how much money the enemy player gets from destroying that building/unit.

The damage stat for units tracks how much health is removed from other units or buildings.

The Number of Units stat is used to show that the images on the map are actually made up of more than one unit, this means that when some of the units die from the health stat decreasing the attack of the unit goes down.

The range stat tracks how far the unit can move in a turn without attacking, if a unit attacks another unit within its range it cannot move for the rest of that turn.

# HCI (Human-Computer Interaction)

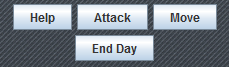
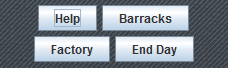
There are many strategy games out on the market at the moment, all of which function with similar, if not the same, human-computer interaction principles. This consists of making use of the mouse and keyboard input devices, along with various buttons on the user interface to aid the functionality.

The mouse input device is strategy games is used for the majority of the interactivity. The left-click of the mouse is used for selection, the right-click of the mouse is used for moving or attacking units, depending on whether the right-click was on an enemy unit, and the movement of the mouse is used to pan the map. The keyboard input devices allows the user to pan the map and zoom in and out of the map, if the functionality is there. The buttons on the user interface of the game itself are used to interact with the functionality of the selection. For example, when a unit is selected, the user will have buttons available for moving and attacking.

These principles haven’t changed. They have been the same with the early strategy games compared to the top strategy games today. Civilization Wars: Advanced is no different. The user will use the left-click of the mouse to select a unit or a building. If a unit was selected with the left-click, the user can right-click in an empty cell to move a unit, since Civilization Wars: Advanced is a grid based strategy game, or the user can right-click on an enemy unit and attack it. The user can pan the map with the movement of the mouse. The user can pan the map left, right, up, down by moving the mouse to the left-most, right-most, up-most, down-most part of the map respectively. As well as using the mouse to pan the map, the user can use the keyboard to pan the map by using the left, right, up, down arrow keys to pan the map left, right, up, down respectively. The keyboard can also be used for zooming in and zooming out of the map to get a wider view of the map. This is done by the + or - keys, which will increment the zoom accordingly.

These two images are from the popular strategy game StarCraft 2. The image on the left is the panel which is shown when you select a unit, showing available orders to the unit. The panel on the right is shown when you want to build structures. Again, this is similar, if not the same, across almost all strategy games and Civilization Wars: Advanced is no exception.

These two images are from Civilization Wars: Advanced. The image on the left is the panel which is shown when you select a unit, where you can choose to attack or move, and the panel of the right is shown when you select a Base building, which you can choose to build a Barracks or a Factory. Each panel has a help button which will show the instructions and the end day button which will end the turn of that player.

At a first glance, is not clear which building is which in the image on the right of the StarCraft 2 panels. This is why we decided to use text rather than images of the buildings as this is more informative to a user that is not familiar with the game. Although a help button isn’t very popular on the panels in strategy games today, we wanted to make sure the user has access to help options whenever available. The same goes for the end day button, this needs to be available at all times. However, StarCraft 2 is a real-time strategy game and therefore, does not need an end day button!

We have tried to follow the interactive conventions of today’s strategy games to make sure that the human-computer interaction principles are intuitive and easily accessible for users that are familiar with strategy games today.

# GUI (Graphical User Interface)

General features on the game UI:

* Time limit

The game is a turn based game. We added a 5 minute time limit onto the game turns. After this amount of time the turn is said to be ended. Meaning that the player is unavailable to make any moves and it switches to the next player.

When it returns to this player again they receive more money and are able to move all units again.

* Day

This is to add some competitive spirit into single player games. The number increments after 1 full day has taken player. A day has taken place when all players have clicked the end turn button once.

* Player Money

Player money is how much you have to build new units and buildings. You gain more money from a) killing an opponent's unit b) killing an opponent’s unit c) when it’s next your turn.

* Current Player

This just lists who’s currently playing the game. This is also shown in the team colour. When on a network game or with UI players this will switch to show it’s currently their move.

* Ally Player

As our game allows for 2v2, this will show who your ally-team is.

* Team Colour

This is a quick reference so you remember who your team colour is. You can only move or edit your own team units and buildings.

Our In Game Context Bar is split up into 3 sections.

**Minimap**

The minimap is an up-to-date view of the whole map showing each of the units, buildings, what you’ve selected and health of units. This allows you to see the bigger picture conveniently and get a broader view of the map at a glance.

# 

**Currently Selected**

This gives details on what you’ve currently selected in the map. Clicking on buildings, units and empty cells yields different outputs which each tell the user something about that item.

Clicking on buildings will show you that buildings current health, there grid co-ordinates and that buildings owner. Clicking on a unit will show its current health, attack damage, movement points remaining, number of units in its squad, its grid co-ordinates and its owner.

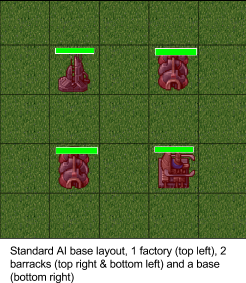
If you click on a terrain it will give you the details of the buffers it gives you (positive or negative) along with what units are allowed to travel on it.

**Button Panel**

This panel changes depending on what you’ve clicked on. For example, if you click on a factory it will give you the option to build a tank. Whereas if you click on a tank it will let you move or attack. The default buttons shown are ‘help’ and ‘end day’. These are there at all times as you should be able to do either whenever you want.

# AI (Artificial Intelligence)

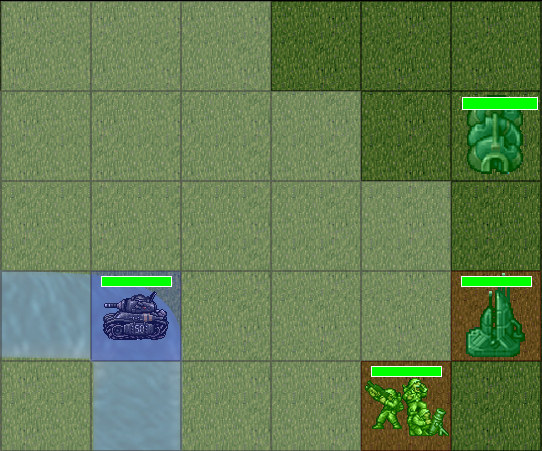
There are three main objectives that that AI attempts to do when it is its turn to play these are to maintain buildings, recruit units and attack enemies. This is the order in which the AI prioritizes its actions, maintaining buildings is top priority because buildings are required to train units and thus maintain an army. Secondary priority goes to unit recruitment so that the AI is always spending money on troops and finally attacking is done last.

When it comes to creating buildings the AI will attempt to maintain a set amount of them. At the start of each turn the number of factories and barracks are checked and if they are below a threshold then they are replaced at the next available opportunity (when money can allow). Each AI player tries to keep 2 barracks and 1 factory in operation so that they can recruit new tanks and marines. The number of each building was decided because of the unit limit on tanks (this being 5) means that a player can quite easily sustain this number through one factory. As marines are effectively unlimited two barracks are enough to create as many as allowed by the players economy. Through testing we concluded that this arrangement was optimal and as such is the strategy that the AI implements.

Unit recruitment is a major part of the game and is something that the AI manages well and therefore effectively maintains and expands its army. Which units the AI recruits is decided by 3 factors. Firstly the amount of money that the AI player has at its disposal, it will check that there is more money available than the cost of the most expensive unit. If it turns out that one particular unit is too expensive for the player at that time then it is not considered. This is also where the AI determines if it has enough money to purchase any units and if not then it will skip to the attacking stage. Secondly the number of limited units on the map is assessed. The current maximum number of tanks allowed by a single player is 5, this is where this is checked and if the maximum number is reached then that unit is removed from consideration by the AI. Thirdly the priority of the individual unit is assessed, tanks are recruited in preference to marines. This is because of their increased strength and durability therefore they are prized higher. Marines are second choice as they are comparably less strong both in attack damage and durability.

Attacking is perhaps the most important aspect of the AI as without it there is no chance of winning the game. Each unit is controlled individually and chooses its own target to attack. This allows the AI to engage multiple targets in one turn which is important when playing a free for all against 3 opponents.

The way that a single unit attacks is as follows, the unit performs a breadth first search in a radial pattern on the map centered at the units position. It looks around it for an enemy unit or building and keeps expanding the search until a unit is found or it exhausts all cells of the map. This search only includes cells that are valid for that units travel, so a marine's search will not take it into a water terrain tile. This means that it only searches tiles that are accessible by that unit, this avoids the situation of a unit selecting an enemy to attack that is unreachable and then getting confused about its route to it.

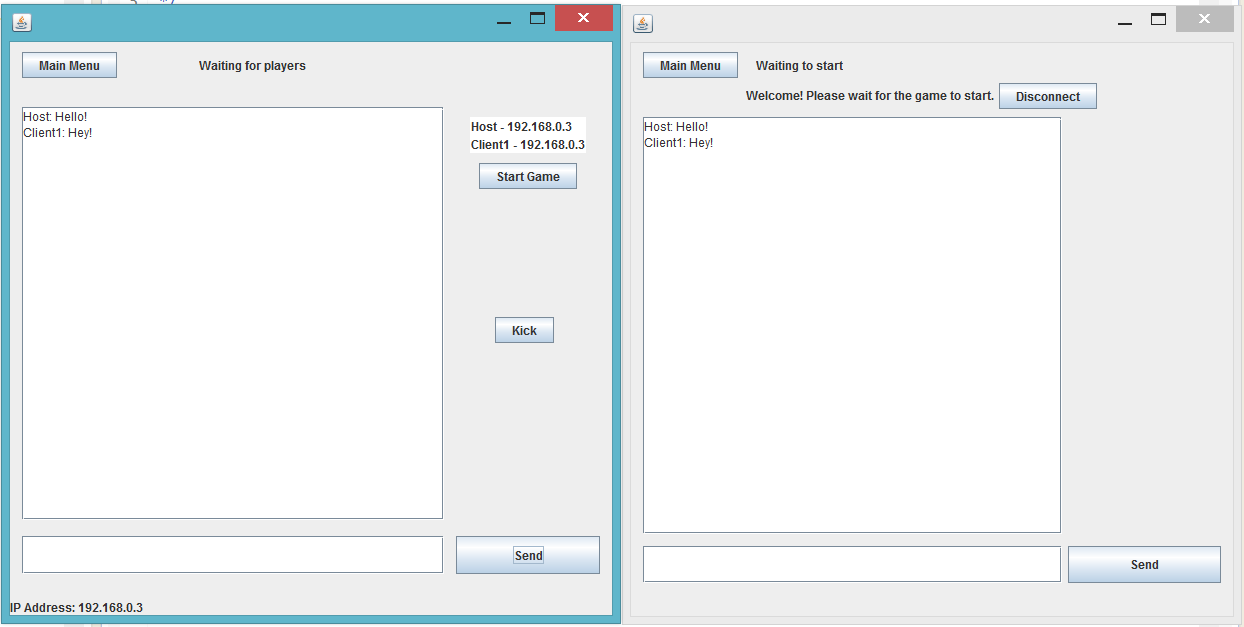
Once an enemy unit is focused as the target a route is plotted from the AI’s unit to the enemys one. The unit then proceeds to advance along this route until one of three states occur. It is possible that the route is blocked (by an allied unit), this can often happen with marines crossing water as the only way for them to do this is via a bridge. If this happens then the unit will move into the closest available cell and cease movement. Another situation is that the unit does not have enough move points to reach the target, if this happens then it just moves as far as it can and then stops. The final situation is that it manages to reach its target. If this happens then the unit attacks the target and then stops movement.

The game has a difficulty setting that gives the AI an advantage over the player. This is implemented through increasing the starting money of the AI player. This means that they have an advantage over the player from the start and as such are able to quickly mass units into an army.

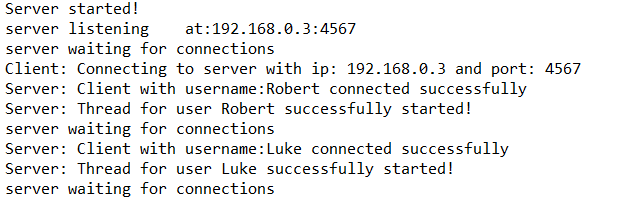
# Networking

One of the features of our game is that it can be played in a local area network. When a player starts the game, besides single player option, he can choose whether to be a host of a game or join an already hosted game. Being a host, you get a screen with a chat area and a list of already connected users. The maximum of user connections is limited to four. Some of the benefits of being a host is that you can choose which map to play on and you have control over who is going to play in your game. You can kick already connected clients and decide when to start the game.

When you join as a user to an already created game, you get in the lobby and wait until the game is full and the host starts the game. Instead of feeling bored, each player can chat with the others in the lobby. Once the start game button is pressed, the game starts on all clients.



Basically, each player in the game is himself a client, who is connected to the host. The server has a list of threads, each for every connected client. It listens for input from all clients and broadcasts it to all other clients. Bigger objects are serialized by Google’s implementation of JSON, GSON. There are four main packets that are being sent, which are for a chat message, for game movement, for game status and for end of turn. The client itself listens for inputs from the host and invokes the appropriate game methods given the object that he has received. All of the communication is done via TCP.

All of the data and processes that are done in the network are tested and tracked real time via console outputs. Debugging is done easily that way.

# Software Engineering

## Class Diagram

The designing of the class diagram (see appendix [item 14.1](#h.xqy19gyymvyo)) helped us greatly during the planning stages of the project to help us understand exactly what was required and how were were going to implement it. Through this we could all see exactly what we were going to implement and how we were going to do it. During the coding of the project the time we spent designing how the classes would be connected and interact with each other meant that we had a very well structured project. This was extremely useful throughout the coding as it made it very easy to place and locate code and because we had thought of how classes would interact it meant that getting access to methods in other classes was easy.

The structure also helped us when expanding the project. Things that we had overlooked in the planning and later came to need were easily implemented into our existing class structure and we could easily grant them access to any objects, classes or methods that they needed.

## 

## 

## 

## User Interface Design

Designing the user interface during the planning meant that it was very straightforward to implement during the coding. Initially we designed the game interface as a group during a team meeting which meant that we all knew how the game was going to look and we had a design to aim towards. 

(Initial drawings from the planning stage of the user interface during a team meeting)

This was essential to do because of the way that we divided up the work. Different people were working on different aspects of the games simultaneously so to have everyone clear about what we were aiming towards helped to keep everyone on the same course of action. Our finished product looks very similar to our initial designs which shows that planning ahead and gaining a good understanding of the project helps keep a project running smoothly.

We applied the same method to the design for the main menus, producing a plan during the specification allowed us to easily implement the designed into the game later on.

## Project Life Cycle

Type of software development life-cycle that we implemented was an incremental agile one. We first had a set of milestones to work towards, these were the core components of the game that were required for it to work. Once these were completed development switched to a more agile approach, with the project changing and evolving as new features were added. There were also unforeseen changes to the project and additions that were not picked up during the planning stages which needed to be implemented.

# Risk Management

Due to the time constraint we faced during development of our project, parts of the final product have been left incomplete. The number of features and functions we wished to implement in a cooperative and competitive game was ambitious, which is why we prioritised the features and functions implemented into the game. Features and functions that the game works without have been left out of development so we could implement the more vital and important features and functions. For example, the development of AI was prioritised over the implementation of Research (a concept in our game in which the player could spend money to power up their units), as the ability to play against an AI allows the user to actually play and win/lose the game instead of the user powering up their units to go attack a stationary enemy.

Alex Bor was unable to carry out work on the user interface due to health reasons. In order to overcome this risk, the task initially for Alex was left out of our initial presentation because at that stage in our project the user interface was not necessary and therefore other aspects of the project was prioritised in order to make sure the project did not get behind schedule.

To make sure that we kept up to date in terms of the implementation of features and functions of the game, we held weekly meetings where team members discussed the progress of their work and any problems and matters that had emerged and we found alternative methods in order to solve those problems. Agendas were produced for every meeting and minutes were taken for when a team member was unable to make a meeting, they referred back to the agenda and minutes to catch up upon what was discussed. Meetings took place as either a face-to-face meeting or using VoIP software: Skype.

# Evaluation

The objective was to create and develop a cooperative and competitive game. The game allows the player to play against AI, choosing from various game types, and play against other human players over a network.

The requirements for our team to work on this project was that all of us were required to know Java from software workshop in the first year at a basic level, however for the more complex tasks we had to use our knowledge of search algorithms from the first year artificial intelligence module or in the case of networking a member of our team had to go and learn the relevant extra Java code expanding on the software system components module from this year.

Our team appreciates the way our project has turned out and we are pleased with the result. All the members of our team have played at least one strategy game and making a comparison to the games we have played and the game we have made we believe we have provided the same experience of the games we have played within our own game, the basic concept of being able to control an army and to have them fight for control over the battlefield was easily established in our game. The only differences between the games we have played and our game is that our game is simpler and lacks the better graphics that other games have.

If there was anything we would could have improved during the development of the project would be the fact that we did leave the more important features till later in the term meaning we had to rush some of our coding or we had to leave out some features, looking back we should have made a better plan to get certain aspects of the project finished earlier so development could have proceeded at a faster rate.

If we had the time we would have tried to add more buildings and units into the game to allow for more variety whilst playing the game. An example of a building we would have included would be a Helipad where the player could build flying units. An example of a unit we would have included would be a Commander which is a more powerful version of the Marines with the ability to destroy non-Base buildings in one attack.

A feature we would have implemented would be to have the buildings and units take a set amount of days to build instead of having the buildings and units being created immediately, this would prolong the gameplay and get the players to think more strategically of when and what to build.

Another feature that we would have implemented if we had the time would be a concept of “Research” which would be accessible from the Base and allows the player to spend resources on creating more powerful units and safer buildings, possibly even creating powerful weapons to protect the Base or supply drops every few days for an extra boost.

If we had the time we would have also created more maps for the players to use as we do have an option to choose the map you play on from within the main menus, this would also add a higher amount of variety in way you play the game when given new obstacles or situations to overcome.

We found networking to be the most difficult feature of the game to implement. Although we did get networking implemented for playing Human vs Human, Human vs Human vs Human or Human vs Human vs Human vs Human, we did not have the time to implement a feature so the user can choose teams of a multiplayer game over a network. It would’ve been great to see the AI be part of a multiplayer game too, so two human players can team up against two AI players. These are some features that would’ve been implemented if we had more time available.

# Teamwork

## Organization

We used several methods to maintain an organized team where everyone knew what they were doing and what was happening elsewhere in the project. Communication was often done daily via skype, here questions and ideas were conveyed through messages left in a group chat. There were also frequent ‘meeting calls’ made when everyone was at home to further discuss what needed to be done for that week and to debate the solution to any problems encountered.

We also had weekly meetings face to face (usually after the meeting with our mentor) where we talked about what was the plan for the next week of work and to take on board the advice from our mentor from that week. These often proved to be more in depth and debate over the best course of action to take was common.

File collaboration was done over two resources, svn for code and google drive for documents. We used SVN to keep the versions of our code synced and upto date with each other as we worked. There were no problems with this throughout the entirety of the project and it was a great aid to us completing our work. The second way we shared and collaborated on work was through google docs. There was a shared folder which we all had access to and through this we could all see the progress of the documents and reports that were were working on. This was a great tool and allowed us to monitor the progress of each other as well as work on a single document all at the same time.

## Problems

We did encounter some problems while working on this project, though we were able to overcome them and continue on as planned. The first problem became apparent during the very first week. The general consensus was that we were going to use Facebook as a means of communicating with each other. Here we could start a group for our team and keep all of our communication private and continually accessible. The problem was that one of our group doesn't own a Facebook account and it was because of this that we ended up using skype as our main communication platform. This actually worked in our favor as the group calls made through skype really helped us later on in the project.

Another problem was trying to find a common time for meetings to take place (both in person and over skype). With five people on varying courses having different lectures and work to complete finding a time that everyone could manage initially proved tricky. This did resolve itself however and we ended up scheduling most face to face meetings after our mentors weekly meeting and skype meetings often changed schedule to fit in with that weeks work.

A third problem was that one of our members was ill for a substantial time at the start of the project. Thankfully his assigned task was user interface and from the gantt chart that we created we knew what other parts of the project did not depend upon this work being completed and so continued to work on those until we were a full team again.

## Teamwork system

We used the teamwork system continually throughout the project however this was usually just to log our work hours for that week. We felt that although it did have other features available for us to use, many of them were excessive for our project and not needed. We also found that the user interface for Teamwork was in some cases not working correctly and there were several instances of it deleting data unintentionally. We did however manage to use it to create a gantt chart for our project.

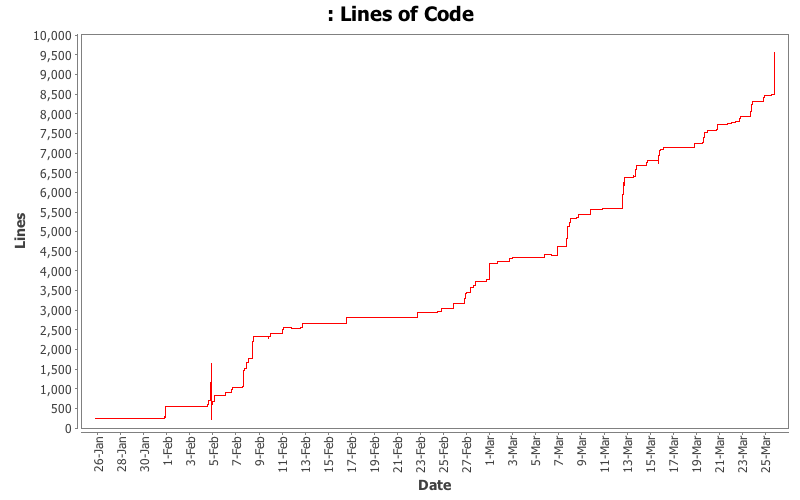
## Gantt-charts

We did create a gantt chart for our project as part of the specification (see appendix [item 14.2](#h.kwbydv7wabsl)). This proved useful in part as it did make us think about what the essential components of our project were so that we had a plan for how to tackle them. It also showed us the dependencies of different stages in the project which was useful to know because (as mentioned above) some parts of the project did not start when we had planned them to.

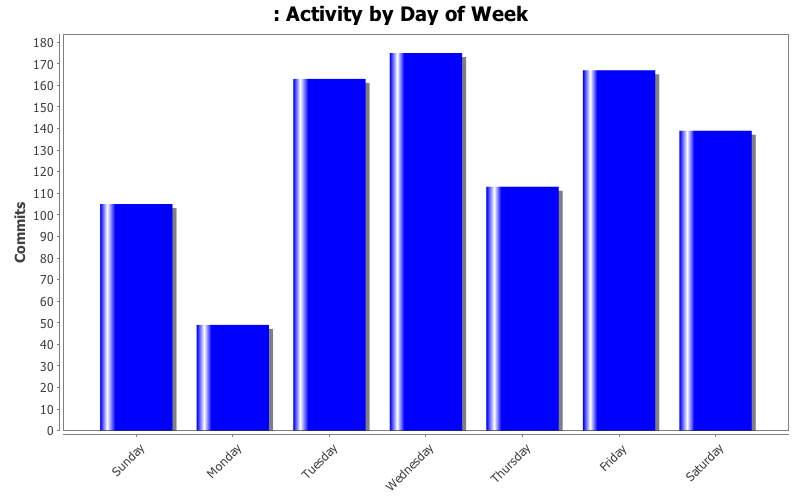
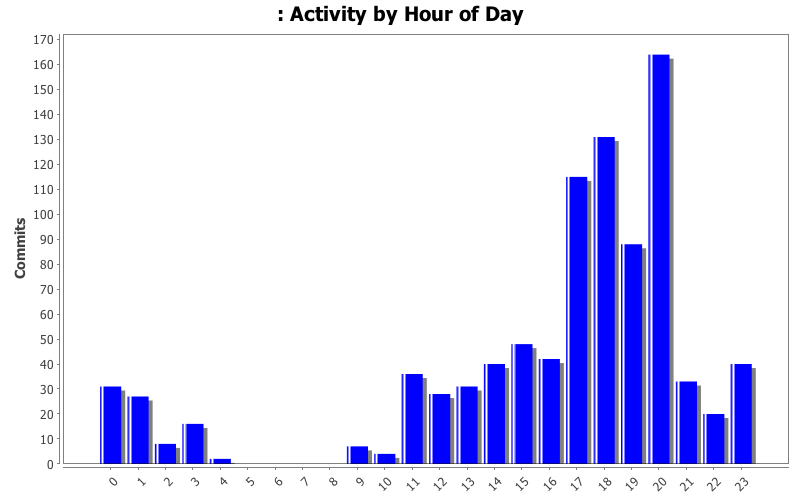
## SVN activity

The SVN repository was how we stored and synced our code between the group members. This was an invaluable resource as it was easy to setup and provided the version control we needed, there were several instances that required us to rollback changes and this would have been difficult without this software. It also allowed us to keep up to date copies of classes that others were working on, this meant that as other classes and methods were completed by other team members they could be used by others.

We used a program called [StatSVN](http://statsvn.org/) to help us analyse the code activity for the past 10 weeks. Below are a few noteable graphs, tables and statistics pulled from the generated report file (these are included for interest). In total there were over 340 code commits and 7800 lines of code.

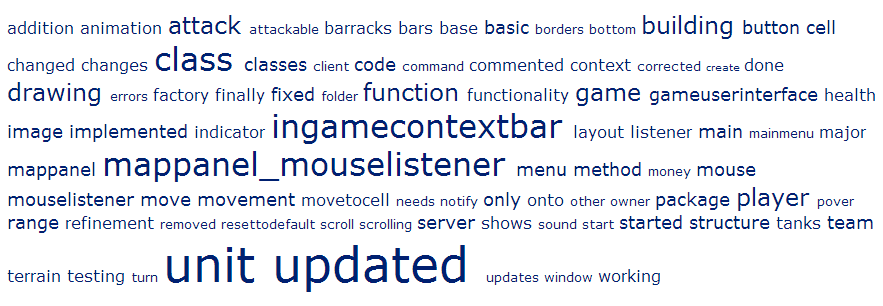


This diagram shows the lines of code produced as the project progresses. As you can see there is a steady gradient throughout the projects duration which shows a fairly regular pattern of work and progress.

These two graphs show an interesting view of when we were most productive. As you can from the ‘activity by hour’ graph see peak times for team members to commit was around early evening with a gradual trail off towards the early hours. From the ‘activity by day’ graph we can see that the spread of work seems to correlate with our timetables. Monday being both a busy day for lectures and the day many assignments are due in for most likely contributed to less work being produced on that day.

|  |  |
| --- | --- |
| **File** | **Lines of Code** |
| TeamProject\_B4/src/userInterface/menus/MainMenu.java | 1065 |
| TeamProject\_B4/src/userInterface/gameInterface/UserInterface\_Bottom.java | 778 |
| TeamProject\_B4/src/map/Map.java | 695 |
| TeamProject\_B4/src/userInterface/gameInterface/MapPanel.java | 679 |
| TeamProject\_B4/src/unit/Unit.java | 590 |
| TeamProject\_B4/src/map/Cell.java | 475 |
| TeamProject\_B4/src/userInterface/gameInterface/MapPanel\_MouseListener.java | 463 |
| TeamProject\_B4/src/game/Game.java | 370 |
| TeamProject\_B4/src/game/Teams.java | 308 |
| TeamProject\_B4/src/player/subclasses/AI.java | 230 |

This table contains the 10 largest files in the project (by lines of code). Many of these would be expected to be in here, for example ‘MapPanel’, ‘Game’ and ‘Map’. However it is interesting that the most code went into creating the user interfaces ‘MainMenu’ and ‘UserInterface\_Bottom’.



Finally this is a ‘Tag Cloud’ of all the words in the commit log messages. More frequent words are indicated with larger letters.

# 

# Summary

Throughout the project we used a range of different tools to make sure that everyone was up to date and knew what they should be working on till the next meeting.

For keeping written documents updated we used Google Drive. We found this useful as it meant that everyone had the most current version of a document and it also allowed everyone to work concurrently on the document in real time - seeing the changes that each person was making.

The downside to this was the constant need for internet access to be able to view these files. It meant that if you wanted to work on something offline or while traveling you’d have to make sure you had updated the file else risk working on something that had changed.

We could have used Dropbox, which also makes version history easy but this doesn’t allow multiple people to edit a document at the same time. Also everyone already had a Google account but not everyone had a Dropbox account. The size of allowed documents is also greater on Google Drive meaning we didn’t have to worry about the size of the files we were sharing with the team.

Teamwork was used for, in theory, scheduling and generating tasks, creating gantt-charts and also keeping a log of the hours we worked. Although it offered many other features most of them were excessive for our needs on this project.

Towards the end of the project it was clear we were no longer using Teamwork to keep track of our project other than the hours we worked per week, as this was assessed. This is down to how difficult it is to actually add information to Teamwork. Considering we used an agile approach to building our project it was always our intent to add or edit features as we went on which isn’t shown through Teamwork.

We could have used basecamp to help manage our team (as do more than 285 thousand people). Basecamp has a clear user interface and makes managing team projects easier.

We used Skype and a Skype group to all stay updated with each other. This was partly down to one of the members not having Facebook. Skype was a useful tool for everyone making sure they were able to remotely contact everyone.

Our team constantly had meetings on Skype or used to or arrange when the next meeting would be on campus.

Skype did sometimes have issues with call quality and also messages getting lost in the chat as there isn’t a way to highlight them. This meant people would also need to be messaged individually if something important was decided on.

Facebook, as mentioned, would have been a good alternative not everyone has a Facebook account. Yammer is also a good service mixing of social network and business into a reviewable state but again lack of people having accounts made this more difficult so Skype was seen to be the best option.

The team found this tools to be very effective in allowing us to work together. They made sure that everyone knew what they were doing as well as keeping the code we had written all merged and up-to-date.

# Individual Reflection

An individual reflection report was carried out by the following members of the team:

* Alex Bor
* Mattie432
* Luke Richardson
* Dhruvil Tank
* Robert Zlatarski

outlining what their overall experience was in participating in this project and what they have learnt from it.

## Alex Bor

In this project I spent most of my time on the design and implementation of the UI context panel. This meant integrating it into each of the other aspects as required making sure it would work as expected. To do this, I needed to research how every other class was working and work with my other team members of how to access the panel and update it.

After this I then went on to add audio, animation and touch up other aspects.

It was a struggle to keep working on this along with all of the other modules and coursework that I also had to complete for deadlines. I found this made it hard to be able to focus on this project as much as I would have liked to. As well as learning new technologies, methods and research.

This did impact on the amount of communication our team had in the middle of our project. Towards the middle it was harder to know what each person should be working on and also organise a time that we would all be free and able to meet. Though this is understandable if this was the only project that we were working on and had no other work I feel we would have accomplished much more. That said we did our best to stay updated with one another and I think we did a good job in doing so.

As a team I think that we all worked well together. We split out the tasks and would keep everyone updated as best we could.

I myself had never used SVN, though I had used Git so I found this to be an advantage and a step-up with learning how SVN worked. That said, implementing it into Eclipse took up a huge amount of time - more than I would like to think it would - as I had never done this before.

All together I feel I learn a lot. My Java skills improved as did my communication skills which I feel was needed and shall help with my current job. This was my first time coding on such a large-scale project using a version history tool and I feel I shall now always be using this in the future.

## Mattie432

I have enjoyed working on this project over the past 12 weeks; from the initial planning stages where we were still deciding on what type of a game to make, right through to coding of the classes and the presentation of the final product. It has been a challenge, with many concepts being new to me (such as SVN and getting my head around how to make a game) but I feel that myself and my team have succeeded in producing a high quality project and a very good game.

I am very pleased with how the game has turned out. It looks good, performs well and has most of the features that we planned at the beginning. It is very robust and even though I have been attempting to find bugs and break it, there has been very little that I could find to do this which is a testament to both our coding and our design. The game looks visually appealing with good graphics and even animation for the units. Playing the game is intuitive and enjoyable with the AI even presenting quite a challenge on the harder difficulty.

Managing time for this project was sometimes difficult as other modules work was a continual distraction also the scale of the project was initially daunting for the timeframe given. The deadlines that we had for the prototype made it easier to get started as there was a fixed time that we needed a working prototype by. In hindsight I feel that it pushed me and the team into being more productive. It was an added bonus that I found this project enjoyable to do as I often spent weekends working on different aspects of it.

This was the first coding project that I have worked on with a team. I found the experience very useful and I will take what I learnt from this into future projects. It has taught me basics such as version control software which I had never used previously, as well as the importance of communication and teamwork. Communication was essential for this project and it would not have been half as good if we hadn't discussed and debated the best course of action to take. I have a greater appreciation for how professional software development teams work and what is expected of individuals in such teams.

Overall, I feel that I have taken a lot from this project. My knowledge of Java has increased, working in a team has set me up with a better understanding of what is expected in future projects and the result we produced is one which I am proud of.

## Luke Richardson

Overall I have enjoyed working on this project due to my interest in heading into the games industry. I particularly enjoyed creating code more than documentation as I found it gave me a better understanding of the project as a whole and an insight into the programming styles of others when our code came together on SVN. The only problems i faced was remembering to log my hours on teamwork as it was a small job done at an inconvenient time and also trying to successfully manage my time.

Speaking about time management in the first few weeks of the project my time management was virtually non-existent it was only when teamwork became accessible that I actually started to plan out my week of work, during those first few weeks time management was not as important as just making sure I was present for team meetings and lectures. When the first presentation was approaching it gave me a reason to start planning time management, even then I didn’t always keep to my schedule, however this did not impact the crucial dates for the project.

Working together as a group was managed easily for us as we each took a part of the project to work on which allowed us to easily update from SVN and commit our code, a significant part of the collaboration was the class diagram our team created in the first few weeks of the term as that provided all the basic information about connectivity within our classes. During times of deadlines some communications were inaccessible due to priority of exercises also communication was split between skype and facebook as some members did not have access to both.

It was hard to maintain meetings outside of the weekly team meetings as some members had other commitments to attend to, eventually as time went on we managed to organise days on which we would get together in the same room and program this made it easier for us to ask questions about the other members’ work and collaborate on the structure of classes. During the time we did not see each communication wasn’t that difficult as any queries we had were put on skype and with us logging onto skype each day it would not be long before someone would respond to the queries we had.

In summary, I found the team project module exciting and I feel I gained a significant amount of experience as to how teamwork could be managed within companies and a better understanding of how to manage my time on projects. I have also expanded my programming knowledge to implement classes and projects in a more efficient process. I believe our final product was implemented to the best of our combined abilities of which I am proud of.

## Dhruvil Tank

During the creation and development of the game, most of my time was spent implementing the interactive aspects of the game and fixing bugs that were found along the way, making changes to other classes to help with the interactivity, paying close attention to HCI principles. I also spent a lot of time testing once features and functionality were said to be implemented and complete.

Creating a cooperative and competitive game while participating in four other modules was not easy by any means. The juggling of coursework for these four modules was a difficult part of the project, meeting deadlines for the project, as well as meeting deadlines for the other modules. As for the end product, I think we have created and developed a great game considering the time we had to create and develop a game while juggling coursework for other modules.

Communication and teamwork is essential in almost all projects, if not all projects. This is something we did not have at times as team members were prioritising work due to upcoming deadlines, which more than understandable, something we all had to do through this project. However, taking part of this project has definitely reinforced the importance of communication and teamwork needed in project, and what the consequences are of if these this is not the case. There are many tools out the to aid with communication and teamwork, something we took advantage and without them, we would by no means have the game we have created and developed.

I have never used software versioning or version control tools such as SVN (Subversion) before, and was not aware of such tools existing. Again, without this tool, we would not have the game we created and developed as SVN helped with collaboration of work done by other team members. There is no doubt that software versioning or version control tools are used in industry to help with work collaboration and to have the experience of using such tools, is something that will be beneficial for when the time comes to work in industry.

Overall, I think it was a great experience to work in a team, creating and developing a cooperative and competitive game while under pressure from other team members and other modules, simulating the experience of working in industry and developing my soft skills and my technical knowledge of Java and various tool such as SVN.

## Robert Zlatarski

I have learned a lot of new things working on the project with my team. Not only did I acquire new programming skills, but also I have gained useful experience in team working. Although I already have a few web projects that I have done in collaboration with a friend of mine, this project was a completely different thing. From the initial planning of the game, until the very end, collaboration with the team has been an easy task. Everyone was accepting new ideas and all of us were ready to implement new stuff in order to create a better product. I really do think that we made the product that we were struggling for.

There were a lot of new concepts that I have faced during the development of the game. One of them was working with a SVN. I have never done that before and at the beginning there were some difficulties synchronizing code and classes with the others. But that got fixed very fast. From the beginning of the development process, I wanted to do the networking part. I thought that this would be a challenging task to be done and I really wanted to learn new things as much as possible. I always enjoy learning from practice. I believe that is the best way to become a good programmer. At first, there were a lot of things that I did not know about making the networking part in a game. So the only way to proceed from that point was to read and watch tutorials and study sockets and already done examples. Then I was ready to start and implement the networking module that would fit our game. Completing that task, made me broaden my networking knowledge that I have acquired during my tuition here at Birmingham. I am really happy that I took that part of the project. I have also mastered some new designing skills, as I have been creating the map image and code.

In the middle of the term, I have faced some difficulties as lectures and homework exercises started to become more and more. I also had to combine my work on the project with studying and working. That made me struggle a little bit and do less work on the project. But once I had all other homework done, I continued working on the project and made good progress during the last three weeks of the term. Communication between my team was done with team meetings and regular Skype conversations and chats. And as collaboration between team members is an important part of the development of a project, ours worked really well.

I am happy with the final product and I think presenting it went really well. I am happy with the work that my teammates have done and I think that we happened to be a really good team.

# Appendix

## Class diagramClass Diagram.png

## Gantt Chart

## Gantt Chart_Page_1.jpg

## Gantt Chart_Page_2.jpg