B5

Project Report

Company of Cowards

B5 (TileBound):

JOE BILLINGSLEY

YU YANG LIN

BEN WESTCOTT

JON DAVIS

PAUL DINES

CONTENTS

1. INTRODUCTION

The brief of this project was to 'implement a game that has at the same time cooperative and competitive aspects'. With such a broad topic there were many possible games that could have been developed to fit the brief. This report will outline the creation of a game called Company of Cowards.

Company of Cowards is a turn based strategy game that supports gameplay with up to four players. The overall goal of the game is to work as a team collecting resources, to create an army which can be sent across a map of mountains, forests, hills, grass and water to destroy enemy bases. The map consists of tiles, with each tile having a specific terrain type and the ability to hold one unit.

Within the game, teamwork is a vital skill to utilise, as players start off with little resources and just a single building. This single building is the player's base, or as it is known in the game, the command centre. If all of a player's command centres are destroyed, that player can no longer participate and the team is left with one less player. It is important for team members to be cooperative when both defending and attacking bases. When defending, teamwork is required to thwart enemy advances and protect each other from being destroyed. When attacking, teamwork is required to help ensure success against tough and heavily defended enemy bases.

The enemy in the game can be made up of either human players connected over a network or artificially intelligent agents. The game allows for these different types of player to be in the same team or pit against each other in various combinations. Whether the enemy is artificial or human, the game mechanics and end goal remain the same.

This report will cover a wide variety of project aspects. It will look at the overall design of the software, overviewing the class structure at both high and low level as well as discussing the general game model as a whole. It will explore the game design in depth and outline the decisions made in terms of the HCI implemented in the game. It will review how the networking works, detailing how it is modelled with the concept of instructions. It will unravel the intricate workings of the artificial intelligence within the game. It will discuss the software engineering techniques and risk management techniques that were used during work on the game. Finally it will examine how the team itself was organised and evaluate the success of the overall project.

2. SOFTWARE DESIGN

2.1 OVERVIEW

The software was designed and built as 6 distinctive modules that could be bolted together; World, Map Generation, AI Techniques, Networking, Game Engine and GUI.

GUI is explained in part 4 of this document.

2.2 MODULES

2.2.1 WORLD MODULE

The World Module is the set of classes that represent the logical world of the game with all actions that can be performed on it. This includes the following classes:

Tile - the class that represents each tile in the world. Each Tile has a position, a type, and works as a container for a Unit.

Unit - the superclass that represents all units in the world. Unit is extended by Army and Building classes. These classes have different functions but share basic Unit attributes. Army Units can move and attack, while Buildings Units can train units and construct more building units.

Sprite - The class for all animations in the world. This class contains a position for the animation, and the queue of Images that compose the animation.

World – This is the class that brings everything together into a single logical World for the Game Engine to access and the game to be played on. It contains a 2D Array of tiles for all Tiles in the map and their respective coordinates as Array indices. It includes an ArrayList of Units for all Units in the world, an ArrayList of all dead units, and a LinkedList Queue for all Sprites to be rendered. Units in the ArrayList, that are alive and on the map, can be accessed by the Tile that contains them. This is a one way relation from Tile to Unit as Units can exist in the world without being on the map itself. This is because dead units are deleted from the main unit list, and temporary "dangling" units are needed for Engine calculations. The class also has all methods necessary to process and access the World and everything in it. To instantiate a World object, a map - a 2D Array of Tile Objects - is required.

2.2.2 MAP GENERATION MODULE

The Map Generation Module is the set of classes that parse and create Tile Arrays from XML files or Auto Generated Maps. It contains the following classes:

MapGeneration - The class that uses a custom Noise Algorithm based on Perlin Noise to generate natural looking terrain with natural height transitions. This class creates and returns a Two-Dimensional Array of Tile Objects.

XmlParser - The class that reads and parses an XML file into a 2D Array of Tiles. This class also includes terrain filtering for water, where water edges are automatically generated to have smooth terrain transitions. This class is capable of parsing any 2D integer array into a Tile array, as well as XML files, which is useful when parsing maps sent over the network.

2.2.3 AI TECHNIQUES MODULE

The AI Techniques Module contains all AI functionality the game will use. This includes Range Calculating and AI Players.

RangeCalculator - The class that calculates different types of range such as movement and vision. Movement and View range calculating is done through A* search on the World, where different Tile types represent different terrain. Each terrain type has a different cost value to move through, which is taken into account by the A* algorithm. This also takes into account any blockages in the world such as other units.

AiPlayer - The class that represents an AI Player and all actions it can perform. This class is also part of the networking module as it extends networking player objects in order to communicate with the server through the Game Engine. It contains all networking functionalities required to perform actions on the world, and send instructions to the server. It also has an integrated priority-based instruction set to play the game. This instruction set will figure out the most efficient way to spend the resources the AiPlayer currently has by giving income priority over production, and by giving army size priority over army cost. The AI instruction set is based on a large army attack strategy.

2.2.4 NETWORKING MODULE

The Networking Module contains all classes required to establish a server-client model for the game. This works by starting a server on the host player's computer, which then allows other players to connect as client players. Although the server is run on the host's computer, the host player communicates to the server as any other Client Player would. This includes AI Players, which join the game as Client Players. The module also involves a parser which encodes and decodes instructions into Strings to send and receive over the network. These instructions are constructed with an OpCode to tell the server what type of instruction it is receiving. The server itself works like a selector by switching over every player connected to it by iterating through a Player Manager. When it receives a message from a player, it sends it off to the other players so everyone is updated with the same data. The server module contains the stages of a Game's life cycle; lobby state, running state, and closed state. This allows the server to be extensible and be used for any other game types given a correctly constructed and integrated Game Engine. The server communication runs on a single thread that reads messages from Players, and sends messages to Player. Each Player also has its own thread pair to send and receive messages.

This module involves the following class groups:

Player - The superclass for all players connecting to the server. This is extended by a large group of Player classes, such as ClientPlayer, AIPlayer and NullPlayer, used to communicate with the Server.

Player Manager - The class that contains all players playing the game. This implements an iterator so Players can be iterated over.

Instruction Builder and Parsing - The set of classes that parse and build instructions to send over the network.

Server States - The set of classes that represent the different states of a game life-cycle. This includes LobbyState, RunningState and ClosedState.

Server - the class that acts as the server itself, which communicates with each player connected to it.

2.2.5 GAME ENGINE MODULE

This is the set of classes that represents the Game itself and all the logic behind it. It includes the Engine with all Game Logic, a Renderer for the Game, and a Controller to play the game. The following classes are included:

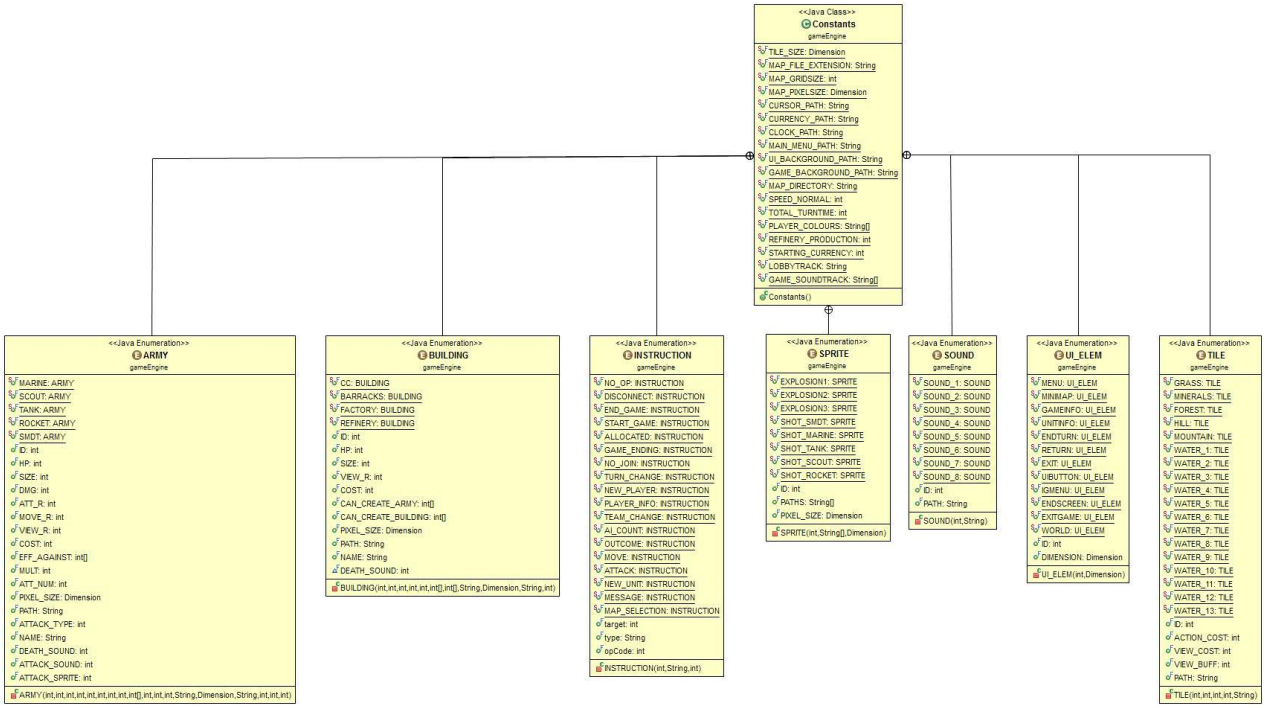
Renderer - The class that renders the World into a BufferedImage. This BufferedImage can then be grabbed and rendered on a JPanel. The BufferedImage can be refreshed by a method called *refresh* in the Renderer. The Renderer draws an Axonometrical Parallel Projection of the World. It is almost completely independent from the game logic itself and only renders what the logical world contains. Methods defined in this class map screen positions to the world, and vice versa, in constant time through linear transformations of coordinate values, which optimizes rendering. The renderer also stores BufferedImages for the map, units, and overlays it has to draw. This allows it to draw the map once, and store it for future use, which speeds up rendering. The renderer also accesses the sprite queue in the world and renders each frame in the animation per refresh of the Renderer.

Engine - The class that deals with all game logic. It deals with all actions a player can perform on the world; attacking, building, moving, etc. This class is also the one that interfaces with the networking. To do all of this, the Engine must contain a World to play the game on, a Player for the player using this class, and a Player Manager for all Players playing the game. The class also contains a Timer that controls the game time and speed. Additionally, it also contains a Tile Object for the Currently Selected Tile which the player uses to perform actions on the game.

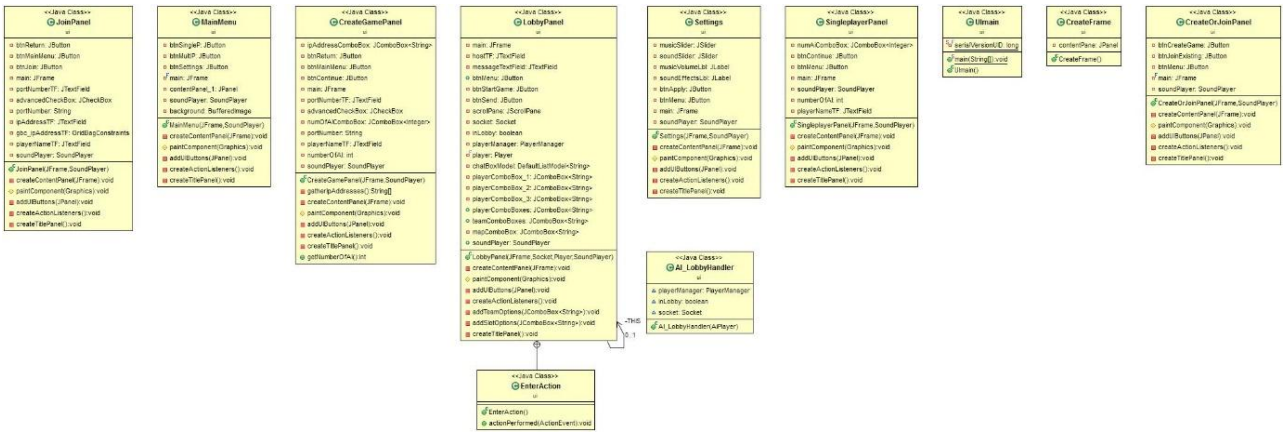
GamePanel - the class that shows the game on screen and allows the user to input actions. This class contains the Renderer, which is refreshes in its PaintComponent method, and the Engine being run by the Player playing this instance of the Game. The GamePanel has custom buttons implemented using BufferedImages and Rectangles to create buttons that can be highlighted and clicked. The GamePanel serves as a Controller that allows the user to call instructions in the Engine according to what the user clicked on.

2.3 LOW LEVEL UML CLASS DIAGRAMS

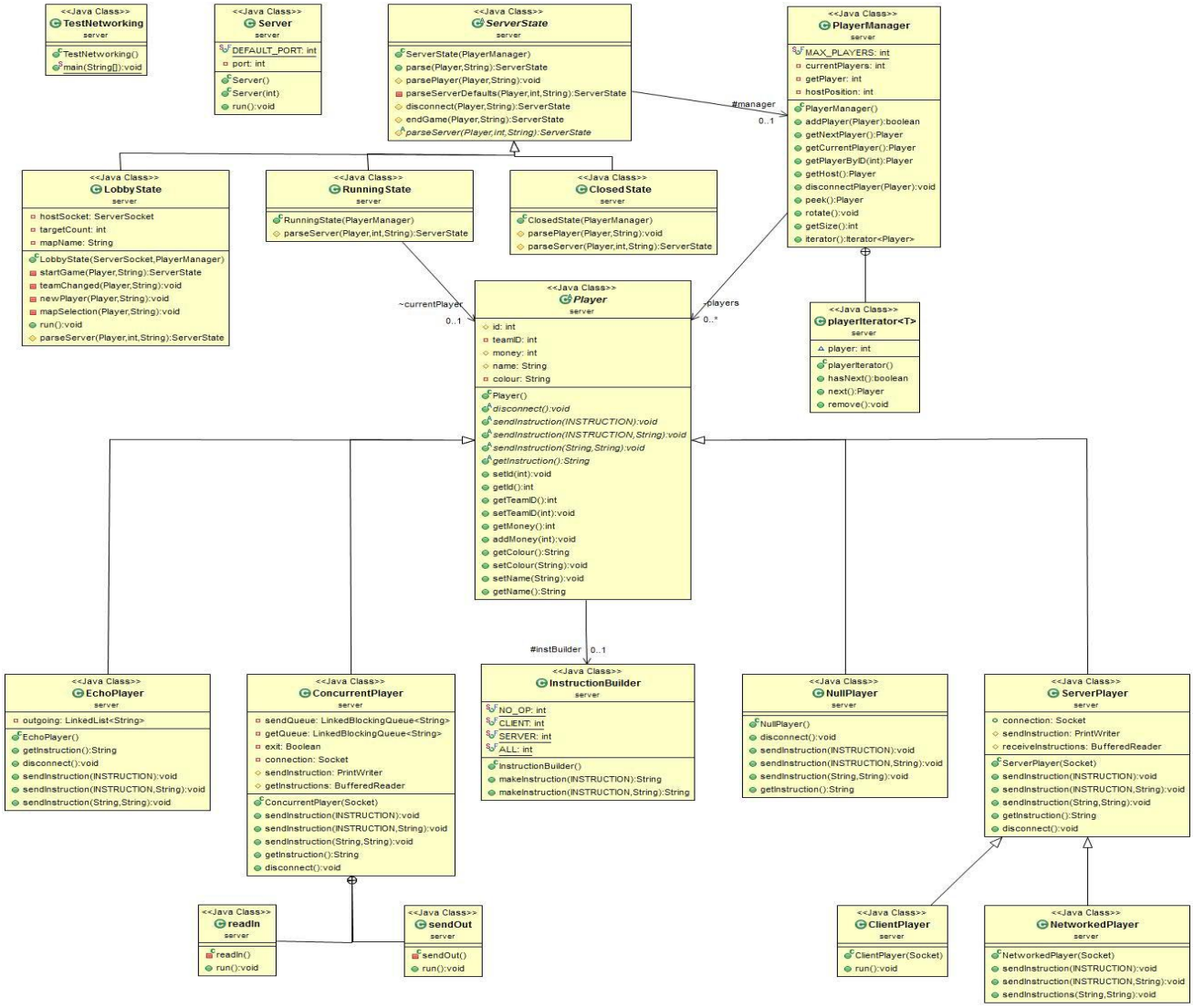
2.3.1 CONSTANTS



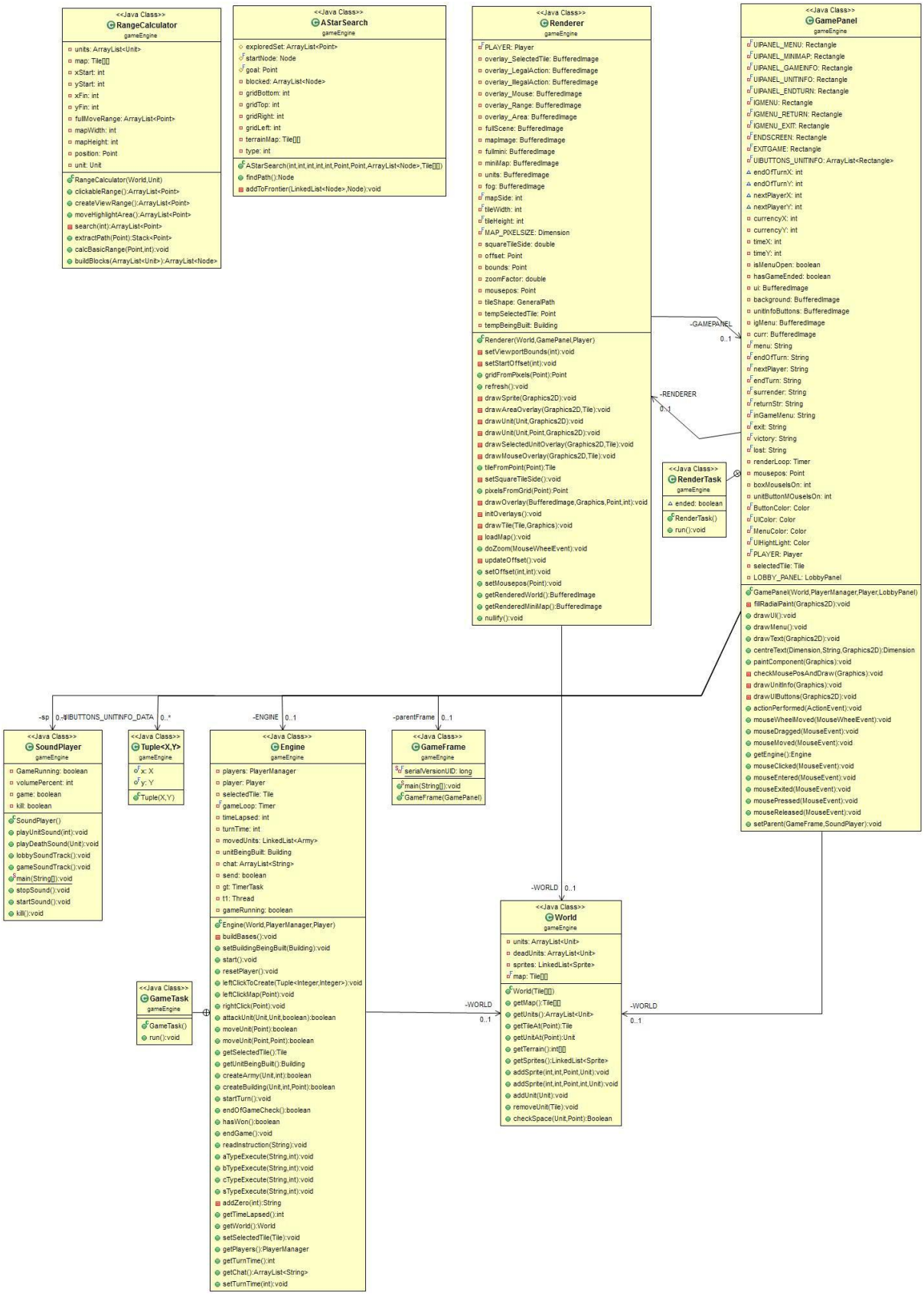
2.3.2 CLIENT UI



2.3.3 NETWORKING



2.3.4 GAME ENGINE/ RENDERER



3. GAME DESIGN & HCI

The implementation of gameplay mechanics, user features and game world, through game design and HCI elements is perhaps the most important aspect of creating a successful game. This fact led to detailed research into how current games, of a similar genre, such as "Starcraft" (*Blizzard Entertainment)* and "Advance Wars" (*Intelligent Systems*, *Nintendo*) handle user interaction and create an immersive and competitive environment with interesting mechanics. This section will highlight how "Company of Cowards" creates its own competitive world, set of rules and intuitive interactions between players and the game.

3.1 WORLD DESIGN

Company of Cowards is set in an axonometric parallel projected world. The world is vibrant, full of lush grass, snowy mountains, dense forests and crystal clear lakes. This world has a powerful and valuable resource known as minerals which is used as a currency. The 4 factions of the world, known only by their colour (red, blue, yellow and pink), seek to acquire as much of the currency as possible, to fight for no apparent reason. This reason becomes even less apparent considering the intelligent life on this world consists of cowards; cowards that would rather run away in the face of danger than engage in large scale combat. It is these strange quirks in "Company of Cowards" which set it apart from other games in the genre.

3.2 GAMEPLAY MECHANICS

Company of Cowards is a turn-based strategy game which allows multiple players and AI to construct bases and deploy armies to attack opposite teams or defend their allies. In order to create buildings and units, the player must acquire minerals, the resource of the game.

3.2.1 TURNS

The turn feature of the game controls all action that occurs during the game and separates it into 2 phases, when it is your turn and when you are waiting for your next turn. The pace of the game is controlled by the turn feature.

On a players turn, they can manage their own base, mineral acquisition and army. It is important to mention a few ways in which the turn timer affects a player on their turn. For example, with a larger army to command, it becomes more difficult for a player to control them and make intelligent decisions both on the battle field and back at their base; this is due to the turn timer having a maximum play time of 90 seconds. It is therefore important for a player to manage their time efficiently. This could mean building a smaller more effective army that can be manoeuvred into strategic position, or massing a large army which is thrown into battle without much thought.

When it isn't a player's turn, they cannot affect their current position in the game. They cannot create units or buildings. They can however, observe the battlefield and see what other players are doing with their own turns. This adds another strategy element of the game. When it isn't your turn, you are actively working out what your next best moves could be to destroy the opposition, much like a game of chess.

3.2.2 RESOURCE MANAGEMENT

Resource management in Company of Cowards involves creating refinery buildings on mineral patches found throughout the game. These refineries are built for a small amount of minerals and can be built from any building as long as a mineral patch is within the building range of the building in question. A single refinery will reward a player with a small trickle of income every time it becomes his turn. It is therefore important to obtain as many refineries as possible, in order to obtain a substantial sum of minerals each turn. However, it is important to balance the building of refineries with the building of structures and army units in order to win against opponents. Simply building many refineries at the start of the game, may end with the opposition's army destroying the player's base before they have built any units at all.

Additional minerals can be found throughout the world, and so players may find it advantageous to expand their influence across the land to build refineries in these areas. This delicate balance between spending minerals to build more refineries and creating army units to attack your opponent is a key skill that players will gradually learn as they play the game.

3.2.3 MOVEMENT AND COMBAT

Each unit in the game has their own mineral cost and set of statistics which affect their movement and combat skill. This variety promotes interesting gameplay tactics. Players may decide to purchase lots of cheap, fast, but low range and low damage units, to rush quickly at the enemy. Others may wish to slowly build up an economy to create large range, slow, devastating units. Additionally, certain units are more effective against other units or buildings, so players can build units that specifically counter their enemy's army and constructions. For example, the scout, a fast, cheap and weak unit, is more effective against refinery buildings, which counters the strategy of quickly building a strong economy.

Movement of units is also largely affected by the map itself. The tiled map contains real obstacles to players, from mountains to forests. Each of these terrain types affect the range in which units can be moved across the map. This mechanic adds further to the strategy of positioning units and constructing diverse armies.

3.2.4 VICTORY

The game is won when all of the command centres of the opposite teams have been destroyed. Victory can be achieved using many different tactics, some of which have been outlined above. As victory is only achieved via this single objective, it is possible to avoid conflict entirely by strategically positioning units and only attacking enemy command centres. This type of victory scenario makes players not only focus on attacking enemy structures, but also defending their own, adding to the overall strategy of the game.

3.3 HCI

Each user interaction in "Company of Cowards" has been rigorously user tested amongst a wide variety of demographics, from computer science students to older users who had no previous experience of games.

3.3.1 CONSISTENCY

User interactions within the game have been tailored to be consistent with other games of this genre. For example, in the game, "left click" lets the user select their units and buildings, when these themselves are "left clicked", they always display an area overlay around them. "Left click"s also allow the user to interact with buttons in the game, such as the buttons to create units. "Right click" on the other hand, lets the user perform actions in the world, such as moving and attacking with units, as well as setting rally points of buildings. This type of user interaction is very similar to "Starcraft" or "Advance Wars", where left and right click have clear differences in their use.

3.3.2 INFORMATIVE FEEDBACK

Feedback to the user has been implemented using both sound and visual cues.

For example, when a unit is clicked, an overlay appears. This tells the user how far the unit can be moved. Indeed, when a user moves the mouse over this overlay, it is indicated to the user where the unit will move to if a "right click" is performed. If the user moves the mouse outside of this indicated area, the colour of the overlay changes, to indicate to the user they cannot move there. When an enemy unit is in range the orange coloured overlay indicates that there is an enemy unit in range that can be attacked. (*See figure 1*).

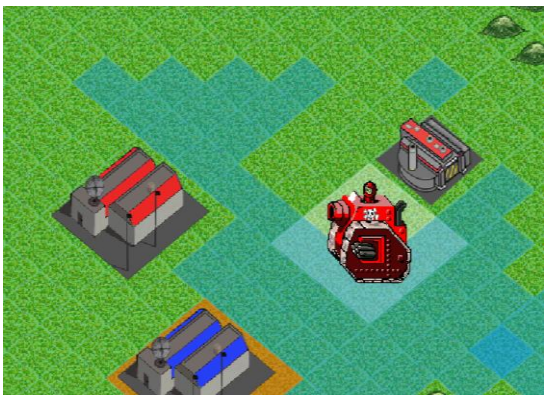


Figure 1

In addition to this, when the user has decided where to move the unit, the unit moves to that position and all players can see this movement. This tells the users that the action has been performed. This process is similar when clicking on a building, except an actual

representation of the building that you are building is displayed when the mouse is moved around the highlighted building area.

The scale of units and buildings is intuitively used to indicate to their respective strength and importance in the game. Larger buildings have more health; larger units are more devastating in combat, while smaller buildings and units are less powerful. While health and damage are displayed to the user via the in game UI, it is not completely necessary to view these, in order to ascertain their significance in the game.

The colour of units is used to show which units belong to a player. A player knows their own colour, because when the game starts, their screen is positioned on their command centre, which has their colour displayed on it. The colours used are distinctive from each other, and so no problems should arise in knowing which units belong to the user. However, if more time was allocated to the project, it would have been beneficial to add a colourblind mode into the game, in order to deal with unit and world colour issues for some potential players.

Sound has also been used to great effect in order to tell the user actions have occurred both in and outside the game. When a game is first initialised, the game music begins before the game has loaded. This tells the user that the game has started and that the rest of the game is being loaded. Another use of sound is in the creation of units and buildings by players and AI. Each time either of these is built a small “pop” noise is played, to indicate that the structure or unit has been created. It tells the user that units or structures are being built in the game, even if they cannot see it happening.

3.3.3 AESTHETICS

The chosen aesthetic of the game is simplistic, yet effective. The art style is cartoon-like but conveys enough information to the user, for them to understand what a unit is capable of, what a building is used for or what a tile contains.

3.3.4 LEARNABILITY AND MEMORABILITY

Through the use of the above HCI techniques and others outlined in the user guide (appendix x), it can be seen that learnability and memorability are important aspects to the game that have been embraced by its design.

The game is easy to learn for a number of reasons. Actions are consistent and follow conventions of similar games in the genre. This means players who are used to playing such a game can pick it up and play immediately. Each action is repeated often during the games gameplay cycle, which reinforces the learning process.

Actions performed by left and right click have a set of conventions within the game. Left click allows the player to perform actions with buttons and buildings, whereas right click is used during the course of the game for moving and attacking with units – actions which physically change the games visual landscape.

Positive reinforcement is one of the main techniques employed by the game to assist in user learning. When an action is performed correctly, a visual or audio cue occurs. For example, when attacking a unit, an explosion animation is played over the unit, and if it has died, it disappears. However, if you do something wrong either nothing happens, you are given an error message (in the case of game client UI) or a visual cue is shown. For example, when a user tries to move a unit outside of its movement range, the user sees that the overlay for moving a unit changes to orange, to indicate that they cannot move the unit.

In terms of the game client UI, tooltips have been used to great effect, to help the user learn what each element does. Although the UI itself has been built to be intuitive on its own, these small tooltips help a user who may be confused about a certain aspect of the UI.

4. GUI / AI / NETWORKING

4.1GUI

4.1.1 CLIENT UI

The client UI is “swing” based and works to guide users through the process of creating either a single or multiplayer game against human or AI players. The UI is efficient in many ways at performing this task, by making sure that minimal input and clicks are needed by the user to start a game. Additionally, it should be noted that all user input is checked to see if it is valid. If it is not valid an appropriate message is displayed to the user, so that they understand what needs to be done, or what they have done wrong.

The settings panel was suggested by users during the user testing phase of the UI development. This was not possible to implement due to time constraints but has been left in to show that we have considered the definite HCI benefit for having a user editable settings panel.

4.1.1.1 TITLE PANEL

This panel is the first part of the game that the user ever sees. It displays the name of the game to user and 3 buttons – Single Player, Multiplayer and Settings respectively. The background of the game is a snapshot of the world which invites the user to explore the game further.

4.1.1.2 SINGLE PLAYER PANEL

This panel allows the player to enter their name into a text field – this text field has a label to show the user that the box is meant to contain the name of the player. The player can also enter the number of AI they wish to play against; up to 3 can be played with or against at once. Tooltips on both components display to the user what input is needed from them. A continue button and main menu button, allows the user to go back to the main menu, or continue onto the next panel – lobby panel.

4.1.1.3 CREATE OR JOIN PANEL

Allows the user to decide to create a new game, or join an existing game that someone has already hosted, via the use of two buttons. There is also another button which allows the user to go back to the main menu.

4.1.1.4 CREATE GAME PANEL

This panel is used by the host of a game to set up various aspects of the game. First of all, they can enter their name into the text field with the label “player name”. The host can choose the number of AI they wish to play against in the number of AI combo box, from 0 to 2 (as 1 other human player is required for multiplayer).

Two of the aims of the UI, are to reduce the amount of time the user has to spend setting up the game and making it easy to accomplish tasks.

As a result, the system filters all the available network interfaces and finds IP addresses which are suitable for players to connect to. These are then displayed in the dropdown box for the player to choose from. This means the player doesn't even need to know what an IP address is to play the game.

Additionally, by default, you cannot change the port number. This is done because many players may not know what a port number is and therefore may not want to change it by accident. However, due to the nature of port numbers, it may be necessary to change the port number. This can be done by ticking the checkbox on the right.

4.1.1.5 LOBBY PANEL

This panel is reached by both the setting up of a single player and multiplayer game. As a host, the player has control of map selection and the teams of the other players playing. Other players, who connect via the "Join existing game option" in the multiplayer panel, are shown in the boxes under the label "players". Players have control of their own team, but the host can change these teams if necessary.

Communication between the host and players is done within the chat box, with messages being able to be typed in below the chat box and sent with the send button (which can also be triggered by pressing enter). For convenience, the hosts IP address is displayed in the top left of the lobby panel, so that players can tell their friends the IP address needed to connect.

4.1.2 IN GAME UI

The in game UI is created using only "awt" graphics and in-house action listeners.

4.1.2.1 MINI MAP

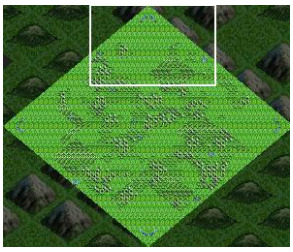


Figure 2

The mini map is displayed in the bottom left of the screen. It shows both the terrain of the map as well as the current position of the screen over the map (indicated by the white rectangle as seen in figure 2). It is possible to zoom on the main map. When this occurs, the rectangle adjusts its size on the mini map accordingly to show how much of the overall map is being shown

Players can click on the map which changes the view on the main map to the area that was clicked on. This allows you to quickly and efficiently move around the map.

4.1.2.2 TURN INFO TAB

The turn info tab contains of all the key information the player needs on their turn. It displays a current turn timer that shows how long the player has left to make their moves. It also displays the next player to play, the current player's mineral count and the overall total game time.

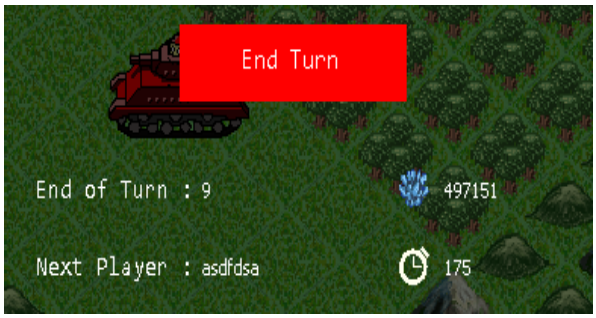


Figure 3

A large end turn button is situated at the top of the turn info tab. When this button is red, it indicates to the player that is their turn (and can be clicked). Once the player has finished their turn they must press the end turn button and they will see that it turns from red to grey to indicate it is no longer their turn (and can no longer be clicked).

4.1.2.3 INFO PANEL (BUILDING SELECTED)

Figure 4

This is the selected building information section, displayed in the bottom left of the screen. When a building is selected it contains the information on what units can be produced. When a user hovers their mouse over a unit or building, it displays its cost and its name to the user. Each box that contains a unit/building, can be clicked on which allows users to build said unit/building if they have enough resources. If they do not have enough resources, the cost text is red.

The health label at the top of the panel displays to the user the health of the building/ unit selected.

4.1.2.4 INFO PANEL (UNIT SELECTED)

This is the selected unit information section, displayed in the bottom right of the screen. It displays all of the information on the unit that is currently selected to the user. A small image of the unit selected is also displayed.

Figure 5

4.2 NETWORKING

The server side of the networking aspect of the game is composed of two critical parts: the 'player manager' and a state and transitions system. Instructions arrive at the server as Strings composed of one character for the 'target' and 'type', two for the 'op code' and an unlimited length for the 'content'. The target is used by the server to determine if the instruction should be handled by itself, sent on to all other players or both. The op code represents the action that the instruction should result in. Content is used for a variety of purposes on the server side ranging from initialising a new player to allowing the host to boot other players. The type is only used by the engine.

All the Server Diagrams are Included in the Appendix B

The server uses the player manager to keep track of all of the players, to determine who the host player is and to respond appropriately. In a normal server cycle the server will first get an instruction from the player it is currently looking at. The instruction will then be broken down into its component parts. If the instruction has a target of number one it will not be parsed by the server and instead sent immediately onto other connected players. The server does this by simply iterating over all connected players and sending the instruction it received back on to all other players apart from the original sender. The original sender must instead receive a 'no-op' instruction as otherwise there is a risk they will skip the subsequent instruction. A target of number two dictates that the server must handle the instruction, whilst number three means the instruction should be examined by players and the server.

4.2.1 INSTRUCTIONS HANDLING

4.2.1.1 PLAYER INSTRUCTIONS

When a player receives an instruction from the server its engine has to parse it and interpret what it should do with it. This is done within the readInstruction method in engine. This method takes the instruction as a String and first checks the second character of the instruction. This second character corresponds to what type of instruction has been sent to the player. Within the game there are four types of instruction; A, B, C, and S. It then reads the next two characters after the type to get the opcode of the Instruction. The opcode is a subtype, and is used to reference different instructions within the four main types. Depending on the instruction type, it calls the appropriate execute method inside engine and sets a Boolean called send to false.

There are four execute methods that each match to a type; aTypeExecute, bTypeExecute, cTypeExecute and sTypeExecute. Inside these execute methods, they all start by checking the opcode of the instruction to distinguish what should be done about the received instruction. For each opcode they parse any extra data sent after the opcode and pass it as parameters to the engine method to perform the correct action within the game. This is

why the send Boolean is needed, as the action methods when called normally by the user through the UI send instructions. We do not want these methods to send instructions as we have the potential to get stuck in an endless loop of receiving and sending instructions. To stop this we only send instructions if send is true.

An example of readInstruction in action is if an attack instruction is received. The instruction is parsed and cTypeExecute is called and passed the opcode and the rest of the instruction. It will read the opcode and match it to the attack opcode. It will then parse the next two characters to get the position of the attacker and the next two characters after that to the position of the victim. It will then get the units at these position and pass them to the attackUnit method to perform the Attack. After the instruction has been dealt with, send will be changed back to true so that the engine can go back to sending instructions normally.

4.3.1.2 SERVER INSTRUCTIONS

Instructions that have been determined to be parsed by the server are then given to the handler that represents the current state of the server. The state-transitions system is used by the server to represent the entire life cycle of a game, from the moment it is started and players begin connecting till the server is closed.

When the server receives an instruction it proceeds to break it down into its op code and content. It may also examine the player who sent the instruction. By examining the op code it can determine the correct action to take. Some instructions such as start game will only execute if they are sent by the host which can be determined by comparing the ID of the player sending the instruction and the ID allotted as host in the player manager. Other instructions such as those directed at players are common to all states and are so defined in the class 'ServerState' which all other states extend. By using a state transition system we build in validation, invalid instructions are automatically ignored, whilst also making it simple to extend the server.

Certain instructions that the server receives are able to move the server into a different state, in particular if the last player disconnects or the host sends an 'End game' the server will move into the 'Closed' state.

After parsing the instruction the server examines its current state. If the server has moved into the 'Closed' state it exits the loop and shuts down as it is no longer required. Multiple servers can be hosted on one computer as long as a different port number for each server. As a result the networking code could be moved onto a dedicated server without requiring much modification.

If the server is not moved into the closed state the cycle will repeat and the server will continue to collect and parse instructions from the players.

4.3 ARTIFICIAL INTELLIGENCE TECHNIQUES

4.3.1 AI PLAYER

The AI player has been set up so that it interfaces directly with the game engine and follows through the same functions that an ordinary human player would go through. As a result the AI player is a fully networked player allowing it to be used both in online play as well as in offline play.

The AI works by parsing the same mouse clicked commands that a normal human player would. As a result the AI itself works as a sort of pilot for an autonomous player. Importantly, it was necessary to have the AI play the game competitively without being too difficult. As a result, firstly, the AI will consider the possibility of building more minerals where ever possible, as this is the key resource in the game. The AI then iterates through the list of available creations prioritising the building of smaller cheaper units before moving onto bigger more expensive units. In the attacking proportion of the AI's turn the AI will directly target the nearest enemy's Command Centre. This will force the human player to produce units and defend constantly until they can gain enough of an army to overwhelm the wave of AI units attacking them. In addition to attacking the enemy's command centre the AI will also attack any of the enemy's units that are within range of its own units.

4.3.2 RANDOM MAP GENERATION

The map generation is done using a random noise generating algorithm. The reason we decided on the use of a noise algorithm was as a result of the pseudo randomness in java not producing realistic terrain maps. The noise algorithm that has been created uses a pseudo random technique which smoothes out the noise over a larger area softening the picture in a similar way to television static at a very close view. This noise is then converted into a height map using the dark areas as low ground and the lighter areas as high. This is then converted into a tile map using ranges that allows us to adjust the height of the map (amplitude) to make the map look more natural.

4.3.3 A STAR SEARCH UNIT RANGE

The units in the game need to be able to both path around blockages and over complex terrain with varying movement values. The algorithm is a common AI technique from the informed search group. The unit pathing works by inputting a goal and a current location. The A star search algorithm will then progress towards the goal testing each possible route in order to determine which is the most direct one and as a result the cheapest to travel. This allows the user to click on the destination square and know that the unit will take the quickest route to that square. This is made possible by the A star search algorithm as it is a complete search technique which guarantees to always find the shortest path.

5 SOFTWARE ENGINEERING AND RISK MANAGEMENT.

5.1 SOFTWARE ENGINEERING TECHNIQUES USED.

From the outset of this project it became clear that a good software engineering technique needed to be established in order to ensure that the overall project could be completed on time and to a high standard. Due to the nature of the game, and the level of abstraction from the completed game, made performing precise top down software engineering techniques like Waterfall almost impossible. Another factor affecting the approach was the time scale, 11 weeks is barely enough to write the game but we had to take into account 2 presentations and the documentation required to tell us where we needed to devote our efforts. As a result of this careful consideration a highly agile software engineering approach was chosen in the form of SCRUM.

The SCRUM technique provided us with the flexibility we required whilst also showing us an iterative style that we could use to accurately assess our progress as the project developed. These key progress points (iterations) gave us a chance to assess how we had got to a certain point and accurately assess how long it would take to perform the next piece of functionality.

5.1.1 FEATURES OF SCRUM WE UTILISED.

Daily SCRUM Meetings:

> Although we were not able to meet daily we were able to meet 4 days a week moving up to 6 days a week in the final stages. During the meetings all members of the teams came prepared with notes on what they had done in the time since the last meeting and these meetings were always held in the computer science common room. We also used these meetings in order to accurately plan what we were going to do in the day's session of pair programming.

Increments:

> We planned our increments to include fully working pieces of functionality before moving on to either another piece of functionality or combining modules. This meant that at as many stages as possible our game was at a working level. This allowed us to both accurately test the effectiveness of new functionality but also allowed us to roll back any changes that moved our game into a non-working state.

Sprints / Sprint Backlogs:

> This is the list of work that has been done to date and also includes a list of all the work that is to be done in the next sprint stage. These can be seen in the documentation provided in Appendix A. We would then complete these sprint stages

in long, often overnight programming sessions, which we would then look back over once the entire sprint stage had been completed.

5.1.2 LIMITATIONS OF SCRUM

Although SCRUM is an efficient and ideal method for us to use in an 11 Week production schedule we also had to take into account that SCRUM does not completely cover all areas of the software development life cycle. For example at the beginning of our project we had to add an initial element of high level design and in-depth requirements analysis in order to produce the project Specification and also to help us define what exactly we wanted our game to do. Looking at such aspects of the project at the beginning of project development is not part of the SCRUM lifecycle.

5.2 RISK MANAGEMENT.

5.2.1 IDENTIFYING RISK

In order to create an effective risk management strategy, it is important to first work out the elements of the project that are going to present the most risk or present the most hindrance if they don't work. On top of this it is necessary to take into account risks that don't come from the task itself such as if the only person who has done the networking was to fall ill, how would that affect the project and would it be possible to complete the project.

To identify risk in the project itself it was decided to look at the connections between different modules of the project and then rank them based on highest to lowest number of connections. This helped us to identify the aspects of the project that had the biggest influence over the overall project success. This highlighted the need to focus on getting the network and the UI correctly integrated first as all other modules communicate directly with these two systems.

5.2.2 MANAGING RISK

Risk	Strategy	Strategy Type
Computer malfunction	All of the code that we write has been backed up on the SVN servers so that the member of the team can continue to work on the CS Lab computers.	Impact Reduction
Personal Illness / issues.	For any given piece of functionality 2 people should understand the code, so that if one becomes unavailable then the other can jump in and continue.	Impact reduction
We are unable to integrate 2 parts of the system.	Routinely test all parts of the system with JUnit tests to make sure they are	Probability reduction

	supplying valid data.	
We run short of time to implement functionality.	Make sure to have a working version of the game before each piece of functionality is added so that we can revert back to a functioning state if it fails to integrate.	Impact Reduction
We are unable to use an external library that we have utilised.	We have decided that we are only going to utilise the java native library and no other external libraries	Risk Avoidance.
Network difficulties during demonstration	During the presentation we will be using our own network router which will ensure that we don't have to rely on the university's questionable wifi.	Risk Transfer.

6. EVALUATION

To evaluate the success of the project, it must be compared to what was stated in the specification.

Looking at the product functions in the specification, we implemented all but one proposed function. The function we do not implement was allowing the user to capture neutral structures. Apart from this, all of the proposed functions were implemented fully so, overall this area is can be evaluated as a success.

The user characteristics proposed in the specification stated that the 'software should be easily learnt' by users of a 'young age' with 'varying computing experience'. During implementation, there was a large focus on HCI and after usability testing with people of varying young ages it can be evaluated that we were successful in this. In-game features and buttons are clearly labelled and the more complex features such as changing the port number are disabled by default.

It is also stated in the specification that 'the system must not be developed with extensive game and UI libraries'. We have followed this constraint and have solely used standard Java to complete our project. This made the project more difficult but the challenge was overcome, with a fully developed game produced at the end of the project.

In regards to the functional requirements, the majority were achieved. An isometric map is displayed, units can be moved, units can be created, the player can attack enemy units, the player can select units, the player can choose to play on pre-made maps or a randomly generated map, and the player can play against people over a network. The player can also play against AI, however they can't distinctly choose the difficulty of AI as proposed. Instead, in the final implementation, difficulty is increased by playing against a larger number of AI.

The non-functional requirements have been similarly successful. In terms of reliability, the game does not 'end due to system error'. The performance of the game is as proposed, with a quick response time from the user performing an action and it being displayed on the screen. As outlined in the specification, there is a slightly higher latency for the other players seeing the action over the network. Availability, maintainability, security and portability are all also as proposed in the specification.

Due to the high percentage of functional and non-functional requirements met by the final implementation, this project can be evaluated to be an overall success.

7. TEAM WORK

7.1 ORGANISATION

Due to the scale of the project, team communication and organisation were imperative. As a team, we communicated with each other regularly across multiple channels. We met in person on the university campus and also talked to each other from our homes using online communication tools such as Facebook and Skype.

Before we started writing any code, we arranged a meeting and discussed what type of game we were going to make and how we were going to work on the project. We decided to do the majority of programming together. We started by meeting on Wednesdays for a whole day of programming, however, as the semester went on we began meeting for programming on Thursdays, Fridays and also on weekends. Programming together came with a number of benefits; it allowed us to know who was working on what and when, it allowed us to work together to overcome complex design ideas and it also allowed us to easily discuss the project as we went.

Pair programming was a common element in our programming sessions. It allowed us to add features that were dependant on two or more modules or solve problems that were being caused by the integration of them. In certain instances which required input from all team members we found it very beneficial to have one person program and everyone examine the code to work out how the class should interface with the other modules.

We did not assign a team leader and therefore all decisions were made democratically as a team. Because we were programming together a lot, ideas and how certain features could be implemented were discussed during these programming sessions. Differing opinions on ideas were debated about positively, with each side being explored and discussed until a conclusion was arrived at as a team.

7.2 TEAM WORK WEBSITE

Due to our continuous communication we had less need for Teamwork's main functionality. As we all programmed together significant bugs were always known by all members of the team. In cases where we worked to solve many bugs in one session we found it easier to write the bugs up on a whiteboard where everyone can see and add to it. This was also more rewarding when a bug was found and removed. Individually we used Teamwork to monitor our hours to make sure we were on target for the week and were not completing less work than our peers.

7.3 SVN ACTIVITY

We used the SVN extensively, managing to reach upwards of 700 commits. SVN was essential to our project as our work on different modules had to be kept updated on all computers. We found it was healthier for the project if we committed fairly frequently,

usually after a new feature was added or a bug was removed. Conflict resolution was a problem further into the project as we integrated the modules. We needed to be careful when editing classes that are shared by multiple modules in case someone else was editing the class at the same time. In particular the constants class which is shared throughout the project sometimes resulted in conflicts.

7.4 PROBLEMS

As with a lot of projects, there were some problems that occurred during development. One of these problems was often organisation when some members of the team had other responsibilities outside the project. To overcome this, we planned ahead. We had a meeting to discuss what tasks needed to be done over the busy period and which tasks would be completed by whom. It was important to have this discussion as confusion could have led to further issues. For example we could have worked on the same tasks as each other or neglected certain tasks thinking someone else was working on it. Over busy periods, we also increased online communication to help each other and discuss ideas.

8. SUMMARY

This document has covered every single aspect of our games development and gives a comprehensive insight into how we approached the task focusing on key aspects such as design, teamwork and HCI techniques.

8.1 SOFTWARE DESIGN.

One of the most fundamental aspects was the software design. This section goes into great detail covering the overall class structure and the fundamental principles upon which the software was created. It also goes into great detail about how the different modules of the software are assembled and how these different modules communicate with each other. It is vital that these techniques were established early on in the project in order to make the integration of modules as smooth and bug free as possible.

8.2 GAME DESIGN & HCI

This is by far the most detailed section of this report that goes into specifics regarding how users are actually going to play the game. It demonstrates every single feature of the user interface and how we think users will interact with them. The game design aspect goes into detail on how we made our game different whilst also having it contain the elements from other turn based strategy games that people are familiar with.

8.3 GUI / AI / NETWORKING

This is quite a large section that goes into detail on some of the more key and required features of our game. It demonstrates some of our implemented graphical user interface designs and goes into detail on our thinking behind those ideas and why we thought they would demonstrate good HCI. The AI section goes into detail on how the basic AI is implemented and also looks at some of the other Artificial Intelligence techniques we have used throughout the project.

8.4 SOFTWARE ENGINEERING AND RISK MANAGEMENT

This section identifies which software engineering technique we based our project on, whilst also touching on how we adapted those software engineering techniques in order to fit our tight time constraints. The risk management section goes into detail about how we identified components of risk and then applied risk management techniques in order to minimise their effects.

9 INDIVIDUAL REFLECTIONS

9.1 PAUL DINES (1243000)

9.1.1 OVERVIEW

I would describe my experience in this Team Project module as an invaluable lesson in the art of adapting to work with a range of people across a complex software project. We first had to establish ourselves as a team before developing a friendship that enabled us to keep working at points in the project where deadlines became stressful. I would say the experience gained from this module matched what you put into your team. A combination of similar work ethics, a passion for computer science and good software design was what enabled our team to turn into a group of friends.

9.1.2 WORKING IN A TEAM

I myself am completely new to the experience of coding with more than one other person and as such I didn't have any idea how my coding style would rank against people of a similar grade. As a result I was apprehensive to begin with as to what people would think of my coding style and how it would work with other more experienced styles. But after meeting my team and getting to know them it was soon apparent that we all had varied styles and our teamwork would be the key to getting our game up and running.

Early on in the project it became apparent that the scale of the project and our restriction on only using native java libraries meant that we would have to have a close and concentrated coding environment. We decided that we would meet three times a week at the start of the project moving up to five times a week in the final four weeks.

As the assignment's deadlines appeared on the horizon and we understood the full scale of what was left to do, every member of our team understood that long hours into the night (often one am) and entire weekends of programming were required. No one objected and everyone attended which is another example of the entire team's outstanding work ethic and their commitment to our task.

9.1.3 SUMMARY

The game we have produced far outstrips what I could produce myself in any time frame and is by far the best piece of software that I can put my name to. I am proud to have been a part of B5 and it has been a pleasure working with every single member. I hope to get the chance to work with this team again and perhaps produce an android version of this game. Times have been stressful and lots of chicken has been eaten to come to this point but I think that we have done an outstanding job.

9.2 JOE BILLINGSLEY (1232400)

9.2.1 OVERVIEW

I strongly believe the module was a great success. We have formed a very strong team and I feel confident in saying we produced a very good game as well. It was challenging, both in the aims of the project and the relatively short time frame we had to create it in. As a result of the module I have also learnt much about networking in Java. The module was by far the most rewarding I have completed and I am very happy to have been able to take it.

9.2.2 WORKING IN A TEAM

The team Tile Bound became is far more capable than any team I have been involved with before. We were passionate about the game we were producing and the game grew better as a result. The long hours spent working on the project together brought us closer together, becoming firm friends as well as a strong team. The entire team were willing to make sacrifices to create the best game we could.

The long hours and hard work we put in allowed us to progress far more rapidly than I could have expected. Our work progressed in leaps and bounds and it was exciting to come in for a team meeting and see how far everyone had progressed. The scale of the project was often daunting though and it wasn't till the last weeks that I really felt as though we were reaching a completed game. If it were not for our careful planning and hard work we would never have been able to create the game in the relatively short time frame we had.

9.2.3 CRITICAL ANALYSIS

We produced some fantastic pieces of work, far more complete than I could have achieved working on my own. Of my personal work I am very happy that the 'PlayerManager' class I devised is a fundamental aspect of the game and server and is consistent, efficient and easy to integrate.

Given more time, the current system for collecting and sending instructions is very good at sending but instructions are sometimes slow to arrive at the server. If we were to spend more time on the project I would implement a concurrent system where instructions are collected by the server on a separate thread and then handled by the server when available.

9.2.4 SUMMARY

This module is by a wide margin the most work I have put into one project and I am very proud of the team we have become and the game we have created. It would be a shame for all of the hard work we have put in forming a tight knit team to go to waste and I hope that our plans to continue working together on other projects are as successful as this project.

9.3 BEN WESTCOTT (1246146)

9.2.1 OVERVIEW

The module has been an incredible experience for me in many different ways. I have always wanted to create a game that I could call my own, and this module has allowed me to do just that. I worked mainly on the UI, HCI elements and art during the project, but also worked in many other different areas of the game. Working on these aspects gave me a great understanding on how users interact with software, and the problems that need to be solved in order to give users a better experience.

9.2.2 WORKING IN A TEAM

Having had previously bad experiences of working in a team when creating software, I was sceptical of how the project would develop. However, I was soon pleasantly surprised that the other team members had a similar work ethic and were keen to work together and help each other to achieve a common goal. There wasn't a day since the project started where we didn't communicate in some way.

I particular enjoyed working with the team during our coding days. We would start with a brain storming session, where we would write what we wanted to implement on that day. We would then work on the separate parts and help each other when any problems were encountered. These sessions were particular useful when we were integrating the separate modules of the game.

9.2.3 SUMMARY

This project as a whole has been a fantastic look at what can be built in a fairly short amount of time, with a willing and hardworking team. Without a doubt, this team has been the best I have ever worked with. We started the project as team members and ended it as a group of friends who are considering further developing the game and releasing an android version.

I have learnt more from my team and their individual coding practices than any lecture could convey.

9.4 JON DAVIS (1252429)

9.4.1 OVERVIEW

This project has been a really rewarding and valuable experience for me. One of the major factors contributing to this is that I was part of a fantastic team. At the start of this semester, I had never worked with any of the members of the team, however as I look back now, I cannot imagine doing the project with anyone else. Due to our complementing personalities and an overall devotion to creating a high quality game, I feel we really grew strong as a team and have actually become good friends. I think proof of this is that the whole team was happy and in fact eager to work on the game in our spare time. Team meet ups increased as the semester went on, with the team even heading into university over weekends to implement extra features.

9.4.2 WORKING IN A TEAM

One of the benefits of working with such as great team is that I personally feel I have been able to learn a lot from everyone. Whilst working together on pieces of code together I was able to see and discuss alternative approaches to different problems. Everyone in the team was always happy to help with any problems hit over the course of the project and I feel I am very much a stronger Computer Scientist than I was at the start of the year.

I would like to work on another project with the team. I think that if we carried on working together as we carried on learning through university we could create some top quality software together. At the time of this report, we are planning on porting the game to Android using libraries such as OpenGL which would be an interesting and exciting thing to do.

9.4.3 SUMMARY

This project has really broadened my view on software development. Before this game, I was very much a solo developer with my only teamwork experience being the creation of small presentations in other modules at university. I was unsure how software development teams actually worked in the real world but now I would feel much more confident to go out into industry and contribute as part of a team.

9.5 YU-YANG LIN (1228863)

9.5.1 OVERVIEW

Throughout this module, I have gained experience in many different fields. It gave me an insight into producing complex software, an insight into the challenges people face when working together; it allowed me to better understand how a group can stay together as a team to produce fantastic pieces of work.

My technical understanding of software development and problem solving has increased substantially. Working on the Game Engine allowed me to experiment and reason about mathematical and logical models; several times I was surprised by the level of abstraction we are capable of reaching. Rendering the logical world on the screen required an organised and efficient solution; we are effectively drawing 3600 buttons that can all be accessed and mapped to in constant time through our own functions. The challenge of doing this for a plane rotated 45 degrees and projected onto the screen helped me grasp the utility of quick and efficient mathematical functions such as basic linear transformations.

This technical insight also gave me critical understanding of industry; I am now able to see games in the market and, to some extent, understand the design, the problems faced, and the reasoning behind the decisions taken.

However, more than any other aspect, the module has very successfully helped me grow as a team member; I had never been able to produce anything of such complexity and quality alone before. Similar to playing in an ensemble, I had to learn the importance of trusting others; understanding the capabilities of others, learning from them, and trusting they know their part and know how to play it well.

9.5.2 WORKING IN A TEAM

I had the luck of being placed in a brilliant team of very capable people; a team of polite and responsible workers that exhibit good work ethic. Each team member has strengths in different areas which proved invaluable for the project.

Work distribution was uniform and we were always available to help each other. Since our first meeting, not a single day has gone past without team communication and planning. We have sent over 5500 messages to each other and the countless hours we have spent together in the CS Study Room are reflected in the 730+ commits on the SVN server.

Through stressful deadlines and difficult problems to solve or fix, the team was able to work strongly together and form a group of friends; which is a very different story to some of the other teams I have seen.

APPENDIX A: GAME DEVELOPMENT

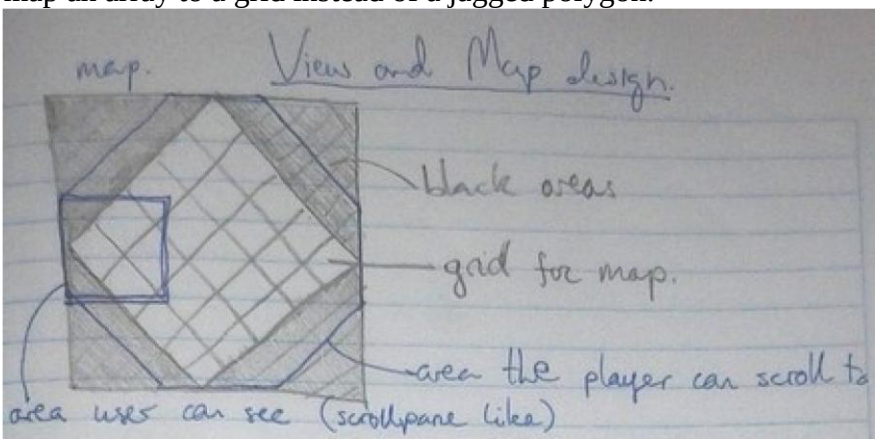
SPRINT BOARD WEEK: 2

We decided on an Incremental Model for the development approach. Waterfall requires well defined requirements, which we don't have as our requirements are left for completion and modification, thus we voted against Waterfall.

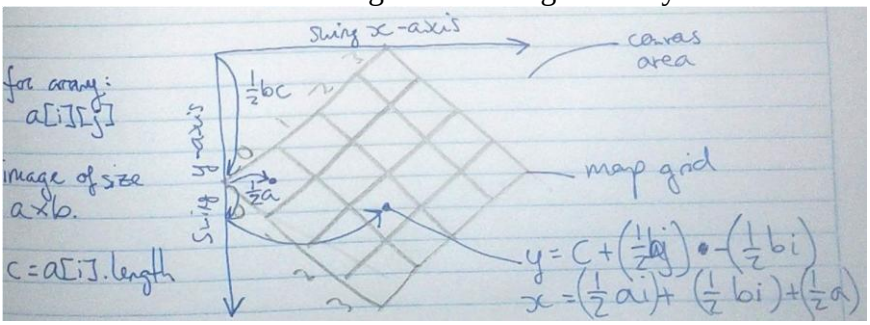
Suggested Design for game during team meeting.

Graphics:

- suggested Flat 2.5D axonometric projection
- asked about simple cartoon art style
- thought about map design in 2.5D isometric view; reached conclusion that the map would be diamond shaped with with everything outside the grid coloured black or some other colour. View of the screen should be like a `scrollpane`. However, scrolling should be limited so players don't scroll outside the map. We considered jagged polygon shape to make the map square. However, it made more sense for a function to map an array to a grid instead of a jagged polygon.



- came up with a function that would calculate the x and y position of the centre of the tile on the swing canvas using the array index values.



Considering an array `a[i][j]` for horizontal values `i` vertical values `j`:

```
a = tile image width b =
tile image height c =
a[i].length y = c + (bj/2)
- (bi/2) x = a/2 + (ai/2) +
(ai/2)
```

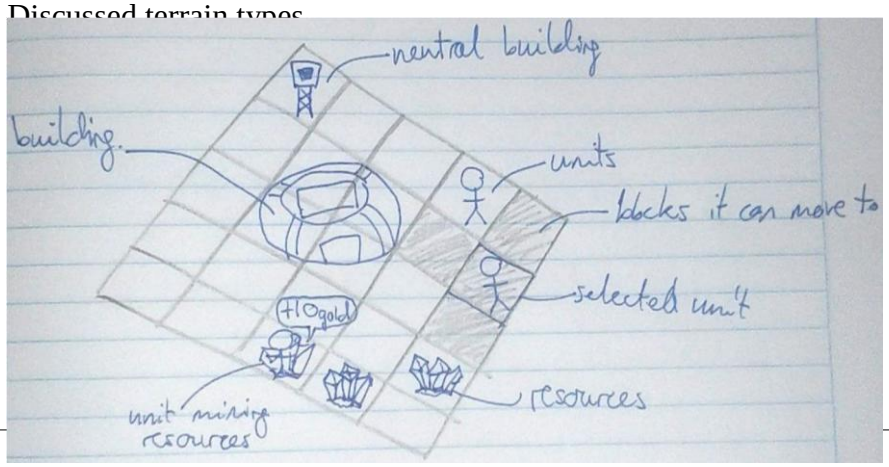
Thu: 10:00 – 14:00
Thu: 18:00 – 22:00

Total: 8 hours

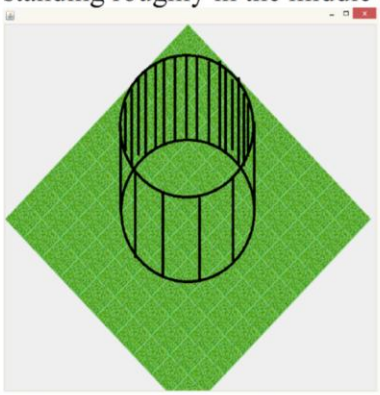
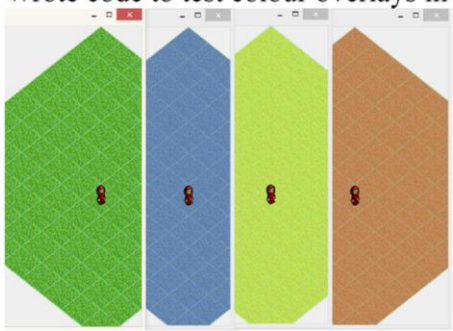
- calculated size the image has to be to fit a 50×50 tile. Discussed whether to use grid outlines on the tile images or to draw them using swing.
- Helped create a test tile and wrote test code for tiling in isometric view

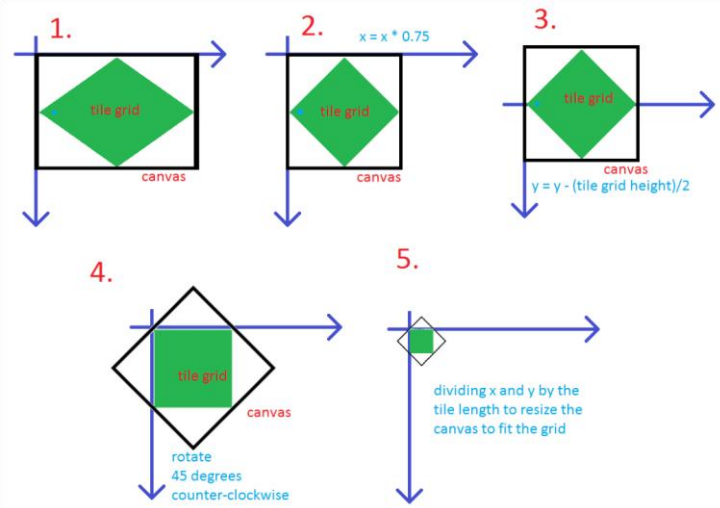
Game Mechanics:

- Suggested worker (harvester/builder) unit.
- Suggested starting base and constructable buildings.
- 2 players per team.
- Discussed units, how much space they occupy on the grid, how they move (Manhattan distance) and types of units.
- Suggested there should be neutral-structures and player-structures.
- Discussed terrain types



SPRINT BOARD WEEK 3

Work Details	Time Spent
Set up SVN on Eclipse to start working on the Project. Uploaded Test_Platform. Discussed UI with Ben and done some work on the Requirements Document. **Graphics:** • Discussed the need for a separate rendering class. • Discussed Graphics with Ben about projection angle. • Wrote a Rendering Platform to test the game rendering, the positioning functions and see what projection angle fits best. • Decided on ratio 4:3 for X:Y for an angle of projection between 40 and 42 degrees. 30 degree isometric looks too flat, 45 doesn't fit perspective properly, 40-42 degrees is the best as it gives us room to draw units approximately in the centre of the tiles. • We chose tile size 71×53, which is approximately 4:3. The sprites should be 71×36, this is because it means that sprites are drawn as if standing roughly in the middle of the tile.  • Wrote code to test colour overlays in Swing.  • Added a GeneralPath polygon to the test Tile Object to know the bounds of the tile and overlay the tile with colour. • Worked out inverse function to get grid position from canvas Point. To do this, I had to map the screen coordinates of the canvas to the grid using transformations. The steps followed were:	Thu: 22:00 – 23:00 Fri: 14:00 – 17:00 Fri: 17:00 – 00:00 Sat: 16:00 – 22:00 Sun: 19:00 – 20:00 Mon: 00:00 – 02:00 Wed: 00:00 – 4:30 Wed: 12:00 – 19:00 Thu: 11:00 – 13:00 Thu: 20:00 – 00:00 Total: 33.5 hours

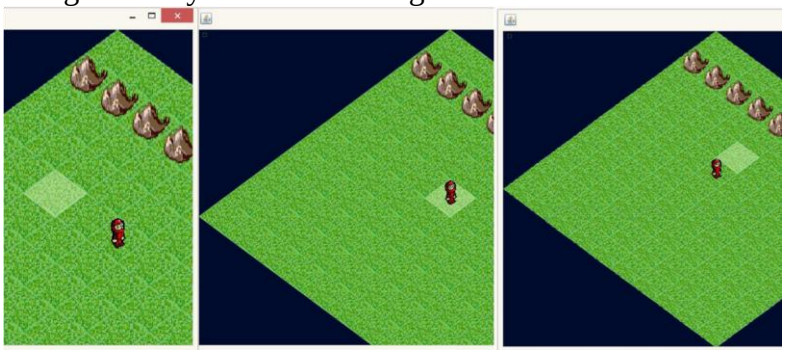


1. scaling the screen to make it equivalent to the 4:3 ratio
of the grid 2. translating the screen vertically to the vertical origin taking into account the grid height.
3. rotating the screen counter-clockwise by 45 degree.
4. scaling the screen axis to match the grid axis by dividing the axis by the side length of a squared tile. 5 . finding the floor of the resulting values.
This is more efficient than finding whether the mouse is contained inside a tile as it is a set of simple arithmetic operations.

- Added Mountain-Tile to the test engine. Added overlay rendering to the test engine for the mouse listener using the Canvas to Grid function.

-

Hacked some code to test zooming. It is ugly but does the job. I noticed that there is a lot of lag for bigger maps. We might have to change the way we are rendering the tiles.



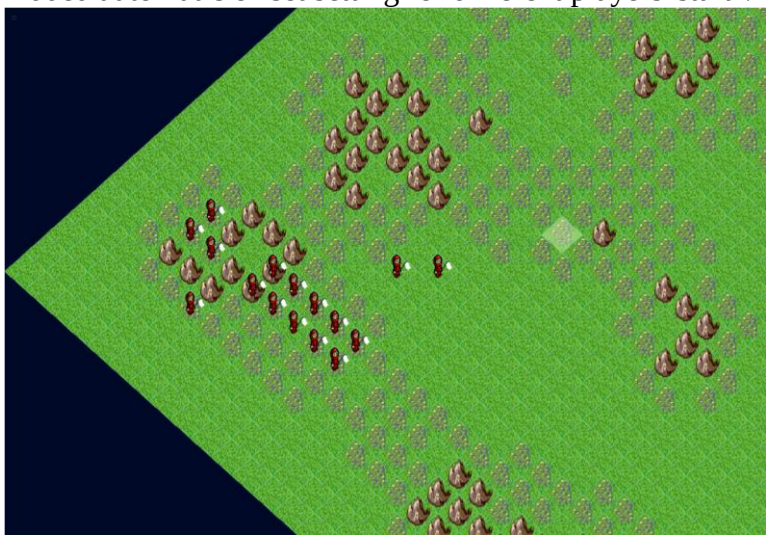
SPRINT BOARD WEEK 4

Work Details	Time Spent
Graphics: • Corrected bounds check at different scaling factors for view port scrolling. • Need to implement mouse relative zooming • Helped Ben designing UI for the Game Engine (not for the client) • Fixed zooming a bit, still not working. • Full screen test working. • To do bounds check using screen size. • Helped Ben with UI design in Swing **Game Engine:** • Game Client runs in frame, once game starts, Game Engine runs in another frame in Full Screen Mode. **Networking:** • Client – Server structure, client has outgoing and incoming queues. Instructions are sent as strings. Instructions are parsed at the server to detect its type. Server distributes the instructions accordingly. • Parser objects written, Instructions objects written. (by Joe) • Server has thread that iterates through each clients' outgoing queue and takes one instruction from each. The moment it takes an instruction, it pushes it into the other clients' incoming queue. **Music:** • Idea: sorting algorithms background (for music) • Idea: military march (auditorium sound bf style) • Idea: retro futuristic music (vintage rock sc terran style) • Idea: orchestrated (Halo, Mario Galaxy style)	Fri: 17:00 – 00:00 Sat: 22:00 – 24:00 Sun: 21:00 – 22:00 Wed: 11:00 – 20:00 Thu: 11:00 – 15:00 Thu: 20:00 – 22:00 Total Hours: 25

SPRINT BOARD WEEK 5

Graphics:

- Discussion on efficient Fog of War rendering. Thought of different ideas such as updating sections of the buffered image.
- Added automatic offset setting for different players' start view.



The above is the view for `player 0`

- Fixed automatic bounds setting and checking for different screen sizes. This is done by identifying the top left corner of the screen in two possible starting settings, horizontal or vertical alignment, and checking for the largest number. This number is then set as the maximum bound where the screen can be with respect to the grid.
- Command Centre building is being made. Helped Ben with the axonometric projection of the building (turned perspective off).

Game Engine:

- Planed Game Engine structure. Decided on the jobs for everyone. The structure of the game engine goes as follows:
  - **Game Engine** Class that manages the running game. It is created when the game starts using the players' data. It manages the game by updating the world, getting data from the world, calling the renderer to render the world, and calling the networking sections for player turns and other functionalities. It will contain:
    - **World** Class: which contains
- Array of **Tile** Objects for the map
- Array of **Unit** Objects for the units on the map
- Array of **Players** playing the game
    - **Renderer** Class which given a World Class and a Graphics object, renders the World on the Graphics object. This will hold all the graphics data, including the required buffered images and the sources to each image file. If any of this fails, an exception should be thrown to alert the game engine and

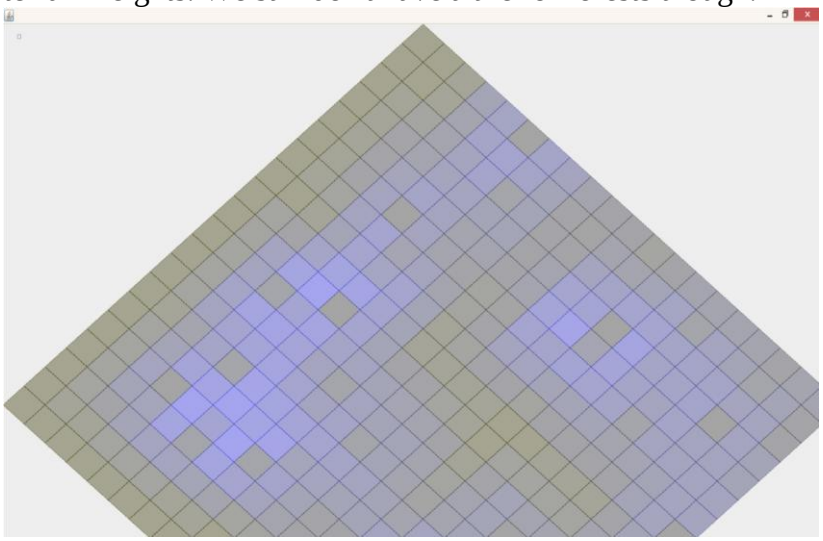
Mon: 18:00 – 20:00
Wed: 11:00 – 20:00
Thur: 11:00 – 15:00 Fri: 21:00 – 23:00

Total Hours: 17

stop it (possibly disconnecting the player).

- **Networking** Section which controls the networking to send and receive data between the players. It should also control the Turns. This includes:
  - A set of **Instructions** that convert instructions to Strings for passing and parse the instructions once they arrive.
  - A **Server** Class that controls the networking and is owned by the user that created the game.
  - A **Client** Class for each player playing, including the owner of the Server, which it treats as a normal Client.
- **A* Search** Class which uses Nodes to calculate movement and vision range.

- A* search was done by Paul. Uses a class for searching and nodes. It shall calculate the area covered by the units vision and movement.
- Map generation was fixed by Paul to have tiles generated at different terrain heights. We still don't have a tile for forests though.



SPRINT BOARD WEEK 6

Work Details	Time Spent
Graphics: • Tested concept for Fog of War rendering. Need to decide on whether to have fog in the model, or in the renderer. This also includes vision. These will be stored as follows: ◦ `boolean fog[][] = new boolean[widthOfGrid][heightOfGrid];` ◦ `HashMap<Unit, ArrayList<Point>> vision;` This is rendered by adding the view points of every unit into the HashMap and removing the fog at those positions. To update this, for the unit being updated, remove the vision for the unit by removing the ArrayList matching the unit in the hashmap. Change the coordinates of the unit, enter the new vision coordinates into the HashMap. To render this efficiently, I will need some way of finding the last unit updated in the model, remove its vision by resetting the fog on it, and reapplying the vision on the updated unit position. Still not sure if you can just modify parts of a buffered image. **Game Engine:** • Will probably need a queue of instructions to perform. • Started creating the GameEngine Engine class that will manage all instructions that modify the world. • Started fixing Unit Class structure • Tried to help Joe fix networking • Kept fixing Unit Class structure. Made a constant class with enums to hold all possible values needed. Deleted sub classes for units, now we have 3 classes: Unit (superclass), Army and Building (subclasses) • Added Tile info to Constants, changed TileObject	Sat: 22:00 – 01:00 Wed: 11:00 – 20:30 Thu: 10:00 – 15:00 Fri: 01:00 – 02:00 Total Hours: 18.5

SPRINT BOARD WEEK 7

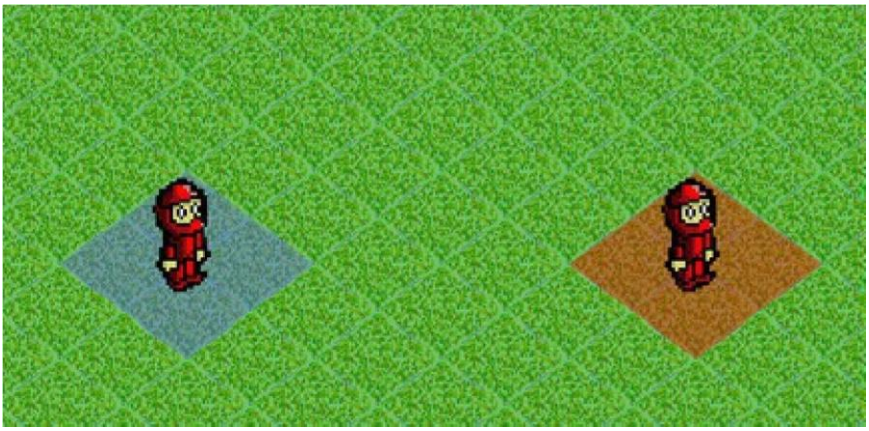
Work Details	Time Spent
Graphics: • Added Forests and Minerals to the test platform.  • Helped Ben with the Hill tile, added it to the test platform. • Paul adds base generation to map generation. Ben adds mountains.  • Started on the Renderer • Draws UI, decided on the bounds checking problems. Decided on structure for the Panel and UI controller. • Got the render to render the world and control it using the game controller (GamePanel). Renderer holds the offset and zoomfactor and can zoom and scroll using the UI controller's mouse listeners.	Sat: 15:00 – 16:00 Sat: 18:00 – 20:00 Sun: 22:00 – 01:00 Wed: 11:00 – 04:00 Thu: 12:00 – 15:00 Thu: 23:00 – 01:00 Total: 28

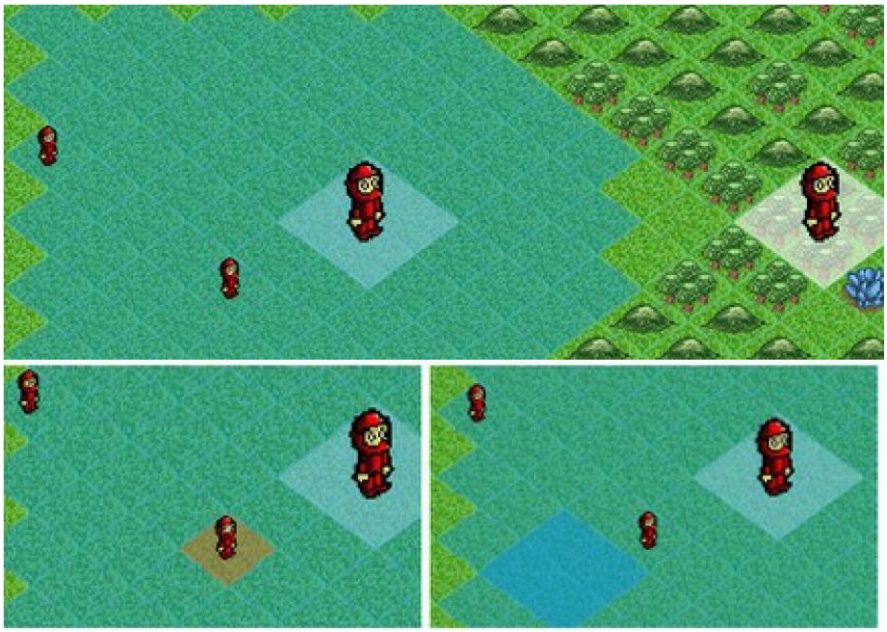
- The GamePanel now holds the rendering loop and the renderer.
- Added minimap and mouse to renderer.

Game Engine:

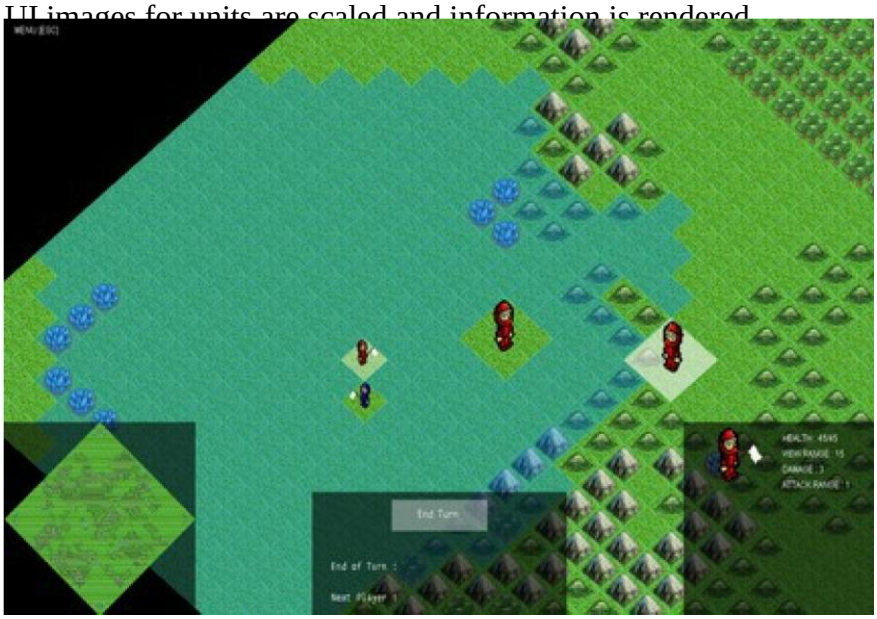
- Added attack method on Army.
- Changed Unit classes. Finished Army and Unit. Added abstract methods on Unit, added more stuff in Army.
- Helped create the movement method
- Helped with GameEngine logic in general
- Paul added an XML parser
- Discussed logic of the methods in the Engine
- Changed structure a bit
- Joe says networking is working.

SPRINT BOARD WEEK 8

Work Details	Time Spent
Graphics: • Added mouse overlays to the renderer. The overlay depends on the selected unit in the game engine. • Discussed possibility of animations • Tested mouse overlays on selected units for range displays.  • Added unit rendering to the renderer. Had to modify unit classes and Constants.  • changed overlays, made area rendering more efficient. • Tested movement instruction rendering • Added bounds checking and click listening to UI • Fixed area overlay scaling • Added left click selection to UI.	Sat: 20:00 – 01:00 Sun: 12:00 – 18:00 Sun: 20:00 – 22:00 Mon: 01:00 – 03:00 Mon: 16:00 – 19:00 Wed: 12:00 – 21:00 Thu: 01:00 – 04:00 Thu: 12:00 – 15:00 Fri: 01:00 – 02:00 Total: 34



- Added movement instruction to the UI, tested rendering of movement.
- Added facing direction to units for rendering while they move.
- Added unit image rendering to the panel UI
- Added colour rendering for units
- UI images for units are scaled and information is rendered

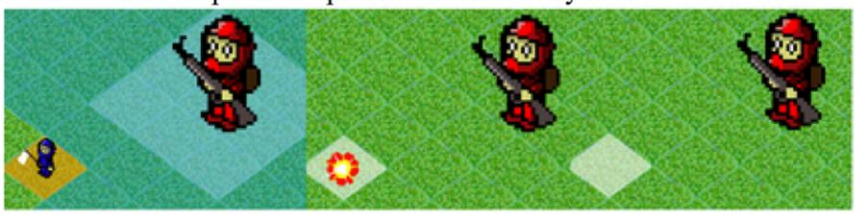
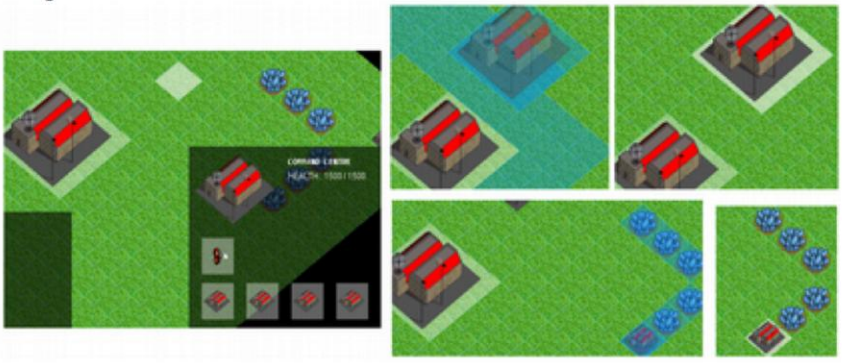


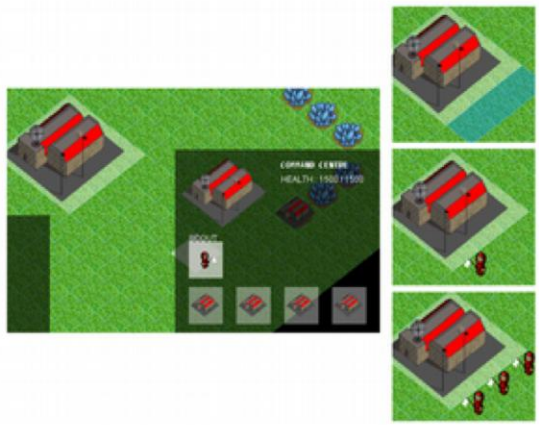
- Added building place holder to test rendering of buildings
- Fixed overlay scaling offset problem
- Fixed unit scaling rendering position
- Helped with calculation for unit info buttons position.

Game Engine:

- Added a point stack for path in Army for the unit to follow when it moves.
- Changed the Engine to start moving units by setting their paths and moving them one square each game tick.
- Fixed ranged calculator class exceptions for index out of bounds and adding null Nodes to the path.

SPRINT BOARD WEEK 9

Work Details	Time Spent
Graphics: • Helped with displaying Unit Info • Added explosions2 • Added rendering overlay for Unit Info buttons • Fixed Building Building Overlay Area • Added Rally Point Gate overlay rendering • Fixed Renderer for build overlays. Had to change logic because references pointed to the same objects. • Added a new explosion sprite that I draw myself.  • Fixed area overlay drawing again. Changed logic and added references to building being created as this one actually changes. • Added illegal operation overlay and unit image overlay for mouse when hovering outside click-able build range. • Fixed area overlay again. Optimized it to draw once. • Display currency, turn time left, time elapsed, on the UI. It is slow. **Game Engine:** • Modified Engine to start working towards unit creation. • Added a Tuple class because we need one • Fixed constant values • Changed Build area to square size instead of view range. • Added logic to set the gates for rally point • Added method to check the surrounding points for building • Helped add build unit method. 	Fri: 12:00 – 18:00 Sun: 01:00 – 02:00 Sun: 04:00 – 05:00 Sun: 07:00 – 09:00 Wed: 12:00 – 21:30 Thu: 12:00 – 20:00 Total: 27.5



- Fixed build unit areas. Had to remove an update range method that was used to remove the filtering done before hand.
- Helped test networking. Lobby with partially working chat system.
- Bug found where if the selected tile is null when a building is selected, the selected building's commands aren't displayed
- Fixed bug with build areas where failed build instructions allowed the user build anywhere on the map.
- Changed constants to add path for currency and clock.
- Added constructor to NullPlayer
- Changed Engine to reset at start of turn.
- Fixed ridiculous bug in Army that reduced the move range while attacking for some devious reason.
-

SPRINT BOARD WEEK 11

Work Details	Time Spent
• Paul created a user manual **Graphics:** • Fixed UI upon ending game **Game Engine:** • Changed how maps are read such that the combo box is automatically filled with all xml files it can find in the directory. • Added loss check. • Added win check in engine. • Added end of game checking. • Added a listener to minimap to set the view • Added win check in AI • port offset • Fixed movement area filtering • Changed start button so it is disabled after starting a game. • Added team check before starting the game • Fixed host team change listener changing all players' teams • Fixed team setting bug so the host can set other people's • Fixed player manager not disconnecting properly. • Fixed server side disconnect which wasn't disconnecting. Thus, fixed server closing. • Fixed bug causing killing marine to end game. • Fixed sound player to kill the thread properly. • Added catch for invalid XML maps • Started on a water terrain transition parser.	Fri: 14:00 – 07:40 Total:

APPENDIX B: SERVER DIAGRAMS.

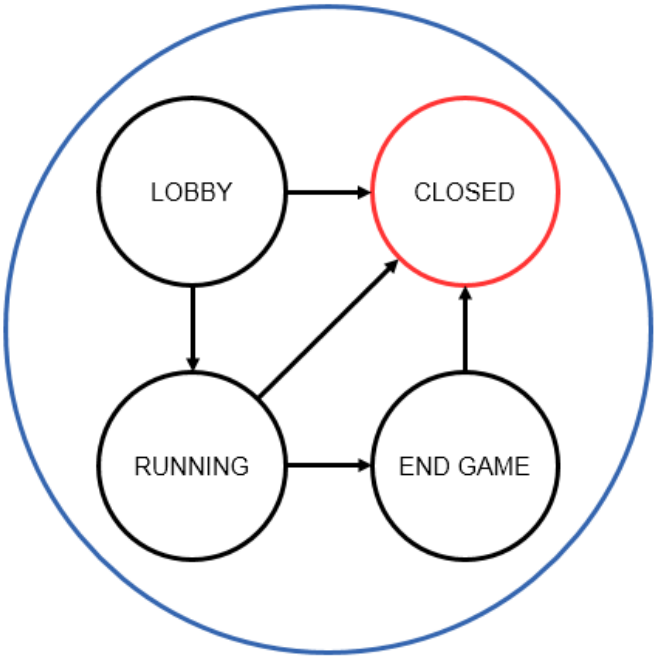


Figure 6 Main Server State

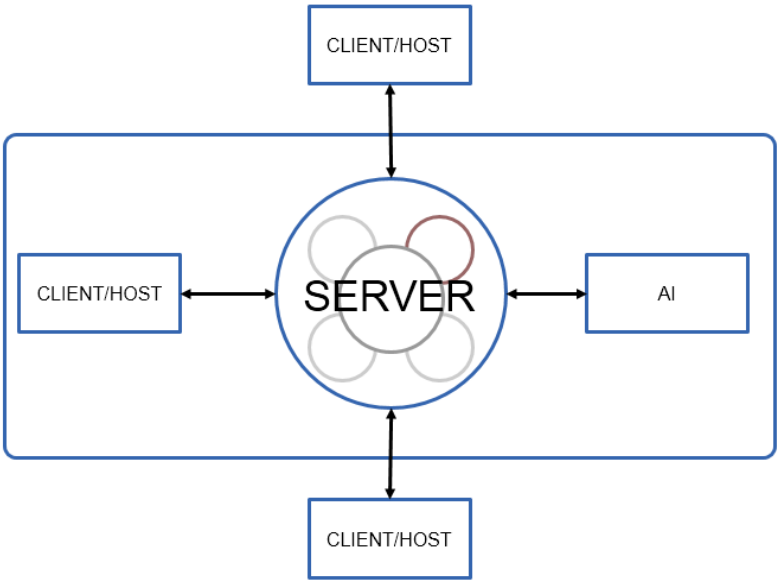
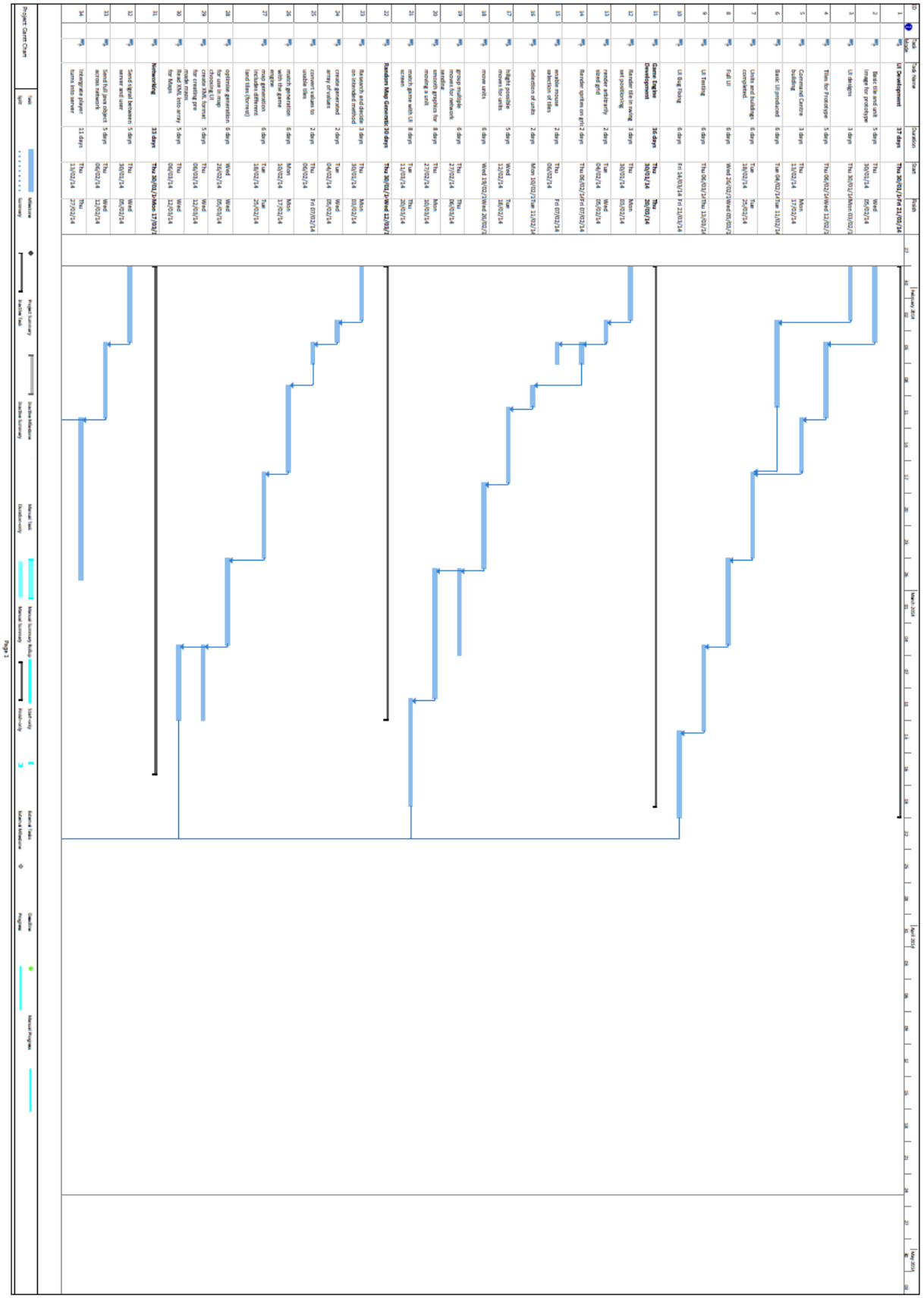
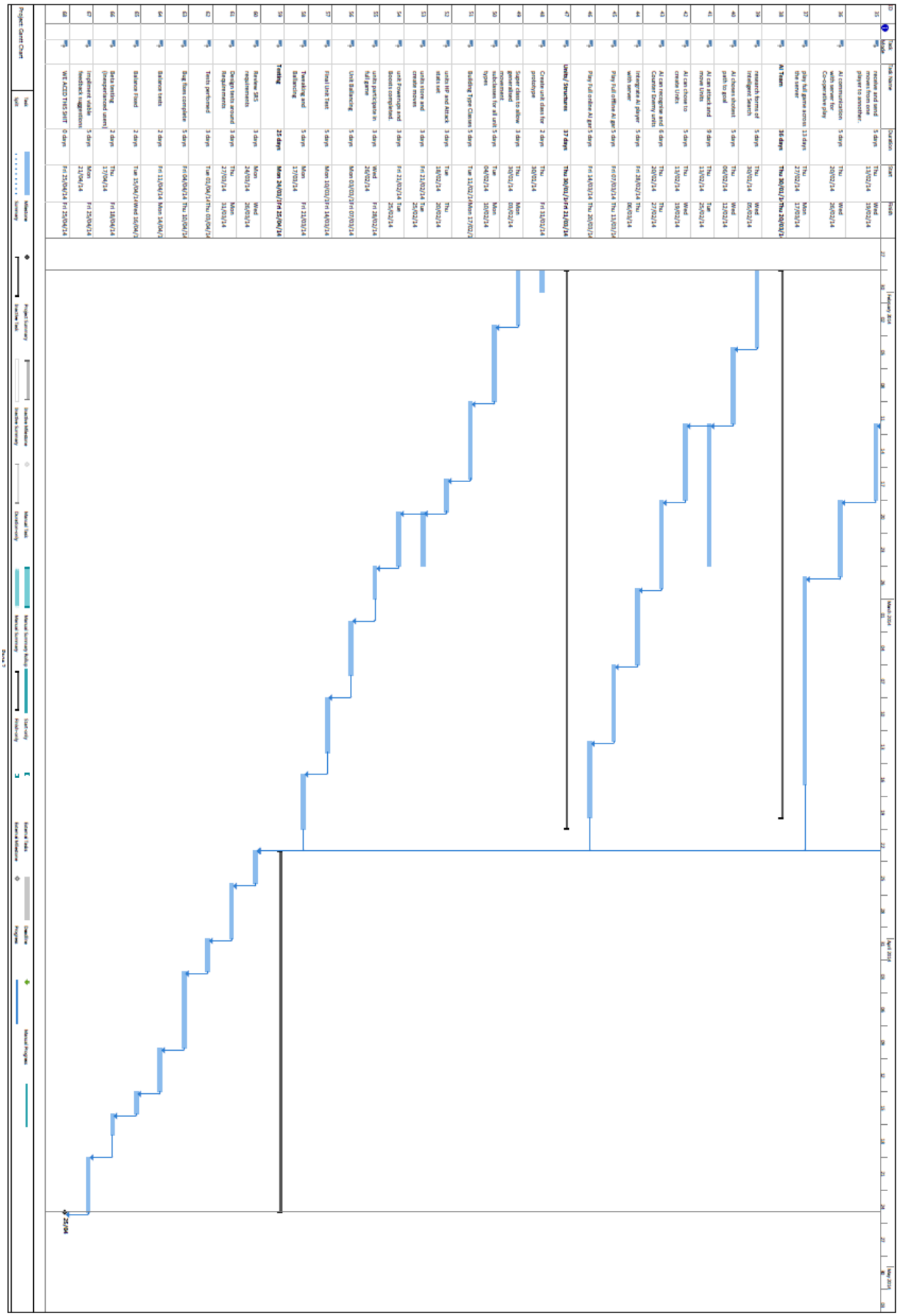


Figure 7 completed server diagram including player manager at the centre

APPENDIX C GANTT CHART

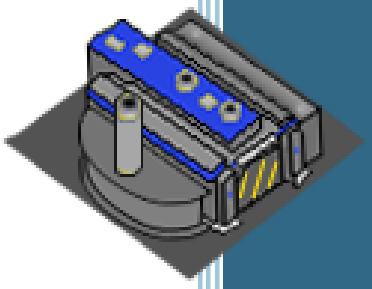




APPENDIX D USER GUIDE

B5

Project Report



B5 (TileBound):

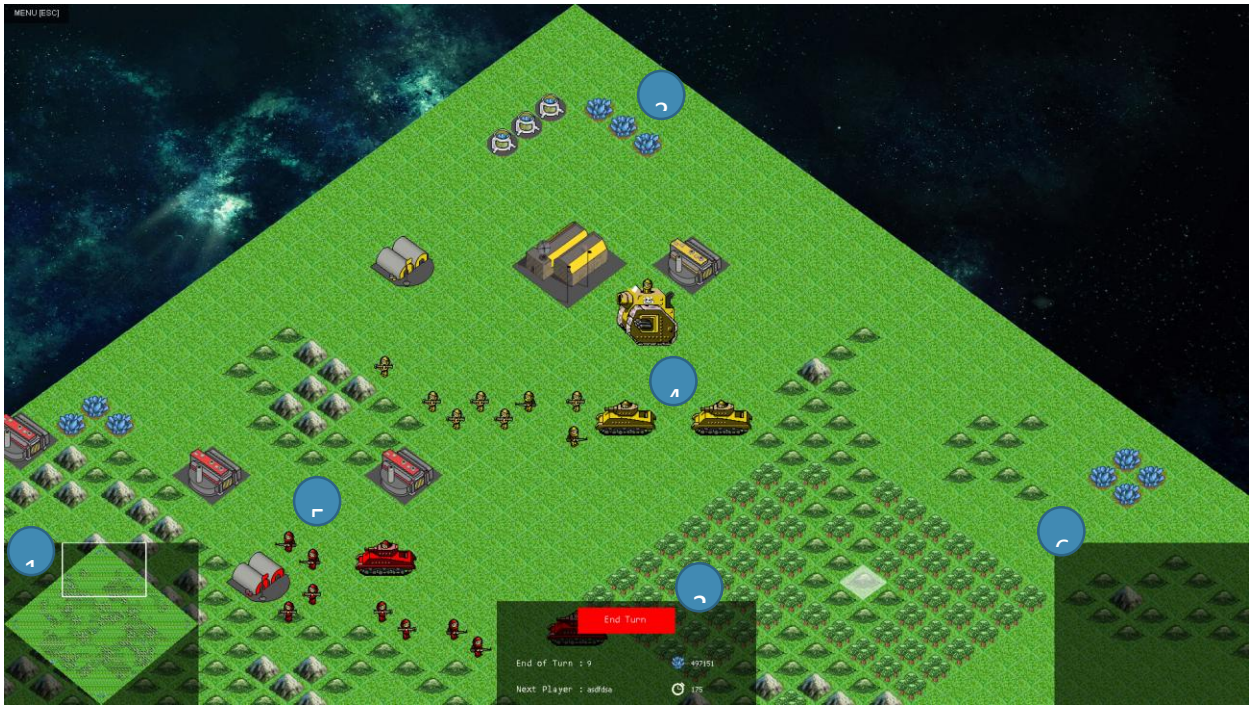
Paul Dines

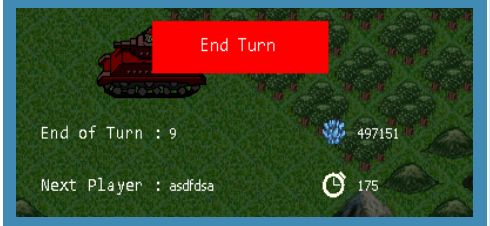
Ben Westcott

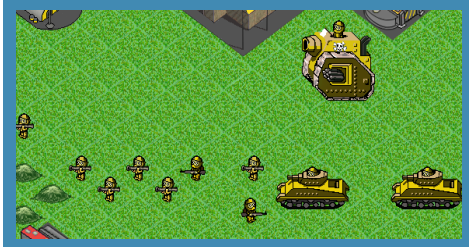
Yu Yang Lin

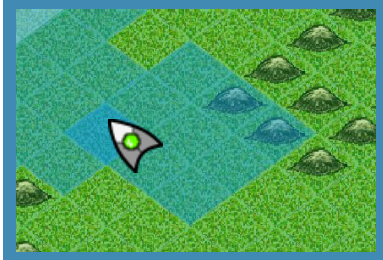
CONTENTS

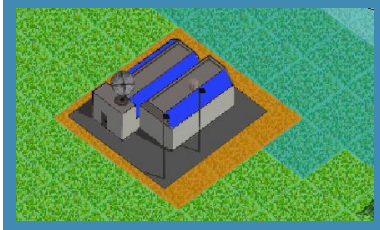
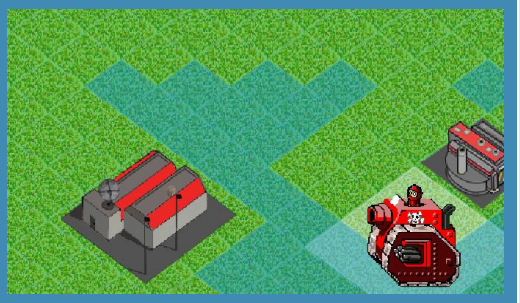
MAIN UI

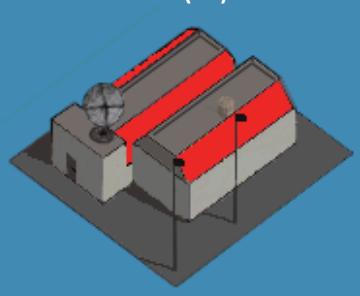
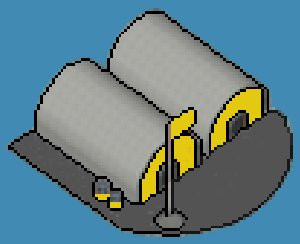


UI Feature	Description	Usage
1. Mini Map 	This is an indicator that shows both the terrain of the map as well as the current position of the screen over the map. It adjusts to zoom level so you are never lost in our game ;).	Changes based on screen position indicates where the current screen is relative to the rest of the map.
2. Turn Info Tab 	The turn info tab contains all of the key information you need for your turn. It has a current turn timer that shows how long you have left to make your moves. It has a indicator for the next player. Your current mineral count and the total game timer.	It is your go when the end turn button is highlighted in red. Once you have finished your turn press the button and you will see that it turns from red to grey to indicate it is no longer your turn. Make sure you have checked everything first as once it's over you can't select anything ;).

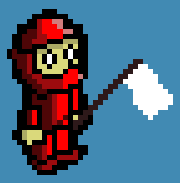
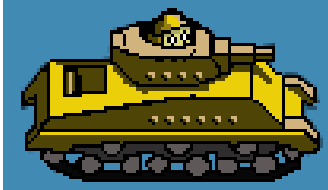
3. Minerals 	MINERALS. This is the life blood of our game if you have enough of them you cannot be stopped. Make sure you build	Build refineries on mineral patches from any structure but make sure the minerals are within range of that structure. Each refinery contributes a whopping 25 minerals a turn so build them fast ;).
4. Your Army 	Your heart, your soul, they will stop you from dying. Just make sure that yours is bigger than your enemies or you might just lose. See units for an in-depth description from each valuable member of your army.	Select your units by left clicking on the unit and you will see the unit information displayed in the unit info tab. You can then move and attack by right clicking over a valid area or enemy unit.
5. The Enemy 	THE ENEMY Kill it violently and quickly before it kills you. If you can, try to kill its command centre in order to win the game.	Left click on your unit to select then right click on the enemy unit in range in order to attack.
6. Info panel (Building selected) 	This is the Selected Unit info sections which displays the info. When a building is selected then it contains the information on what units can be produced.	Hover your mouse over the unit you want to construct in order to see its cost and the name of the unit. The health at the top of the panel displays the health of the building/ unit selected.
7. Info panel (Unit Selected) 	This causes the information panel to display all of the information on the unit that is currently selected. Make sure to check this so you don't attack with a nearly dead army.	Contains information on the unit including unit health damage attack range and importantly the number of attacks that are left for the current turn.

UI Feature	Description	Usage
1. Currently Selected unit 	The currently selected unit can be identified by the clear white highlight across the entire occupied space of the unit.	Make sure to check all of your units to identify which of your units are running low on attacks and health.
2. Unit Final Position Highlight 	When you have selected a unit it would be useful for the larger units if you could see where they will end up after the move. This handy highlight shows under your cursor after a unit is selected and shows you just that.	Left click on a unit to select them and you will see the movement range highlighted, you will then notice that if your mouse is inside the highlighted area then this final location will be clearly visible.

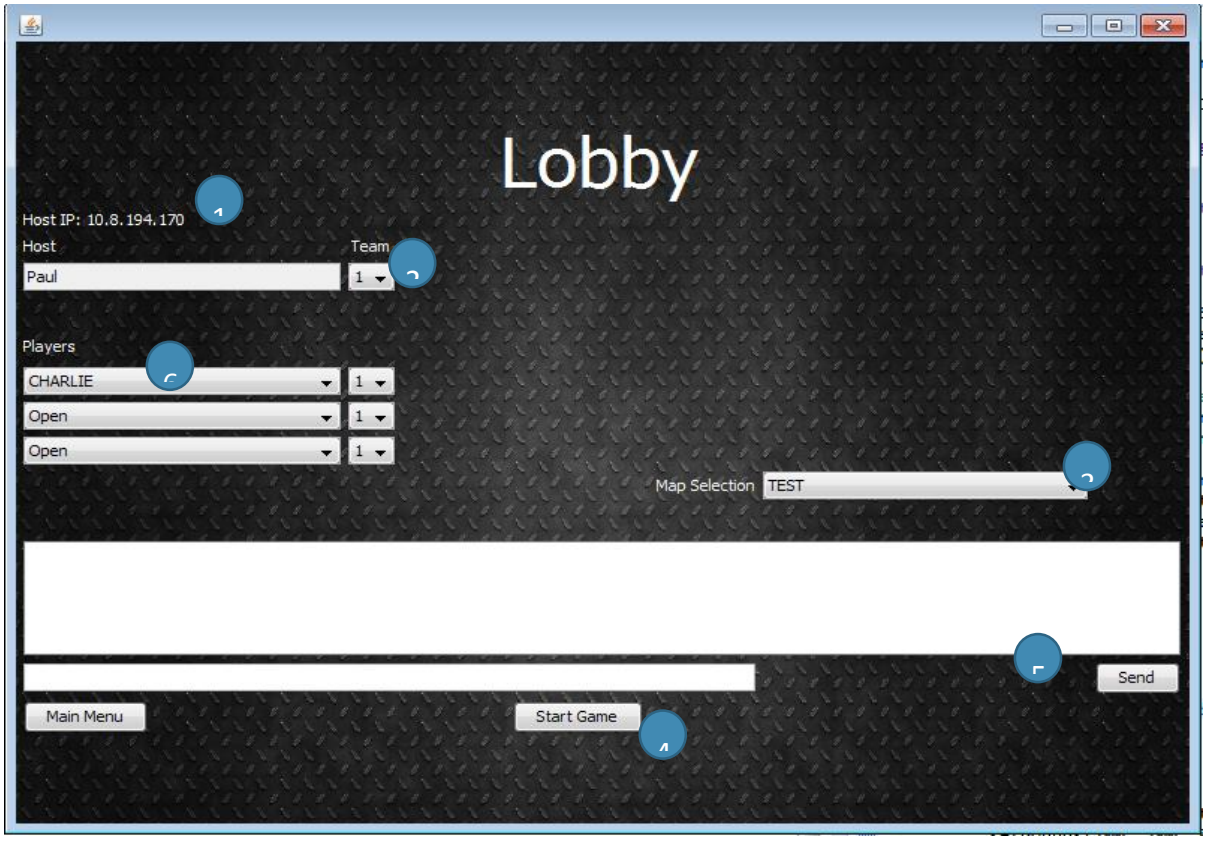
3. Enemy in Range Highlight 	Here you can see the enemy building has been highlighted in a red/orange box. This is a handy little indication that the unit is in range and ready to be violently destroyed.	Left click on your unit. If your unit is within the attack range (displayed on unit info) then the enemy will be highlighted in red indicating that you can now right click on the enemy to shoot at them.
4. Unit movement range Highlight. 	This is the highlight for the entire range that the unit can possibly move in one turn. This movement range will decrease as the unit moves until it does not exist at this point the unit will have run out of moves.	Left click on your unit to select it. If the unit has available moves left then a large blue highlighted area will appear. Right click with your cursor in the blue area to move the unit to where your cursor is clicking.

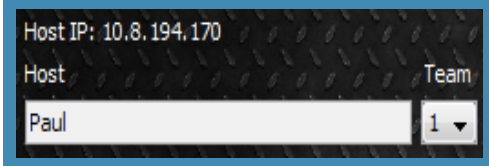
Unit	Cost	Damage	Health	Number of Attacks (Per Turn)	Movement Range	Attack Range	Traits
Command Centre (CC) 		999	5	Scout 	800		The biggest badest building in the game. You get one at the start and if you lose it then you lose the game. You can build more but they are quite expensive and I bet you would prefer to build something else ;). Protect it at all cost but it can take one hell of a beating.
Refinery 		50	5	NONE	50		The way to make money in this game is to put as many of these on as many mineral patches as possible. Each one generates 25 minerals a turn to expand to build more and win the game, but be careful they are very weak and easily destroyed.
Factory 		250	5	Tank  SMDT 	400		This building produces some of the more scary units in the game but never underestimate the power of a tank built right next to the enemy's base. So expand, expand, expand.
Barracks 		150	5	Marine  Anti-Tank 	350		Barracks produces the other two people units of the game and are relatively cheap to produce so make them on mass. This is the cheapest building to produce (Bar refinery) so use it to expand your map control and build more minerals.

BUILDING STATS/ DESCRIPTION

Scout 	30	3	35	3	11	1	A very weak unit that has a large movement range, it is useful for blocking opponent's minerals and being annoying. It has a melee attack so get in quick and fast.
Marine 	50	10	45	3	9	5	The first Combat unit it is cheap and effective in large numbers. Good vs other light infantry but annihilated by armour.
Anti-Tank (ROCKET) 	125	15	100	2	7	6	Cheap way to combat tanks strong vs armour but watch out for the SMDT as nothing can beat that.
Tank 	225	20	160	2	7	6	The first basic armour unit is good against marines but due to its limited number of attacks can get easily swarmed. Good unit for taking our CC's
SMDT (Super Mega Death Tank) 	999	150	800	5	6	10	Without a doubt the best unit in the game. Almost impossible to kill and even harder to afford. Expand well and you may be able to purchase your enemy's destruction.

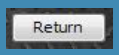
ULTIPLAYER LOBBY



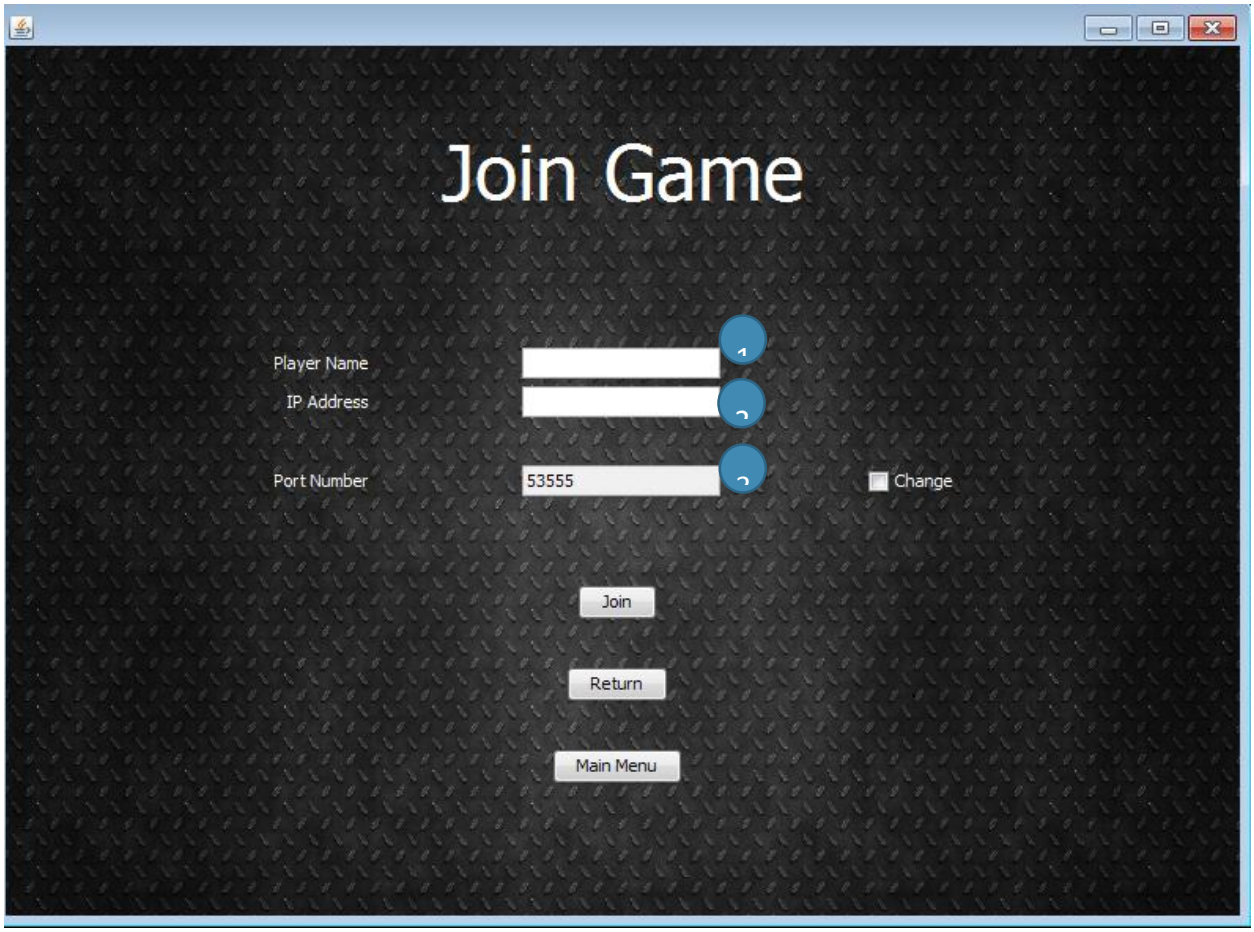
Lobby Panel Elements	Description	Usage
1. Host information 	This part of panel contains the information relating to the host of the game that is needed in order to allow other people to join the game.	When a host sets up a game he needs to make sure that the people joining the game know the ip address that is displayed on the top left.
2. Team Allocation 	These check boxes allow all of the players to change their teams. The host can change all aspects of the game but the individual players can only change their own team	Click on the drop down box and choose from 1 of 4 possible teams. People on the same team will not be able to attack each other and if one player on their team loses their CC then the entire team has lost.

3. Map Selection 	The map selection function allows you to choose from a range of pre made maps with the additional option to choose a random map that will be randomly generated using a noise algorithm giving a unique map for each play.	Select the drop down box and choose the desired map. If you want to have a randomly generated map then select the RANDOM map option in the dropdown box.
4. Start Game 	LET IT BEGIN	Once all players are in and ready click this button to begin the carnage.
5. Lobby Chat 	This handy little panel allows you to chat to your palls in the lobby. Perhaps you can smack talk your way to victory.	Enter your text into the box on the bottom left and press the enter key or the send button. Your message and others messages will then display in the above box.
6. Players 	This handy side bar contains all the players that have joined the lobby. You can also see any of the AI players that have been added, but watch out that AI is aggressive.	When a new player joins the lobby they will be automatically added to this list.

Create Lobby Panel Elements	Description	Usage
1. Player Name 	Player name. This will be the name of the host in the lobby	Type your name into the text field and try to keep it PG.
2. AI selection 	Here is where you can select the number of AI in your game. It is capped at 2 as you are only allowed 4 players a game including you and your friend.	Select the drop down and choose between 0,1 and 2 AIs
3. IP address. 	This is the hosts IP address that is needed for all of the other players to connect	The Game automatically detects your IP address and then gives you the option to keep the standard IP and chose the IP V 6

		version if you prefer. These are automatically entered into this text box.
4. Port Number 	This is the port in the computer that the pc will use in order to interface with the game server.	This is set by default to the value of 53555 which is a port for java but can be changed by advanced users but selecting the change option to the right of the box.
5. Continue 	Once all of your settings are as you want them you can continue to the lobby panel	Click the continue button to progress to the lobby panel.
6. Return 	Moves you back one screen in the menu system	Click the return and you will be taken to the multiplayer join or create panel.
7. Main Menu 	This exits all the way back to the first menu screen to make exiting the menus quicker.	Click the Main menu button and you will b returned to the first menu screen.

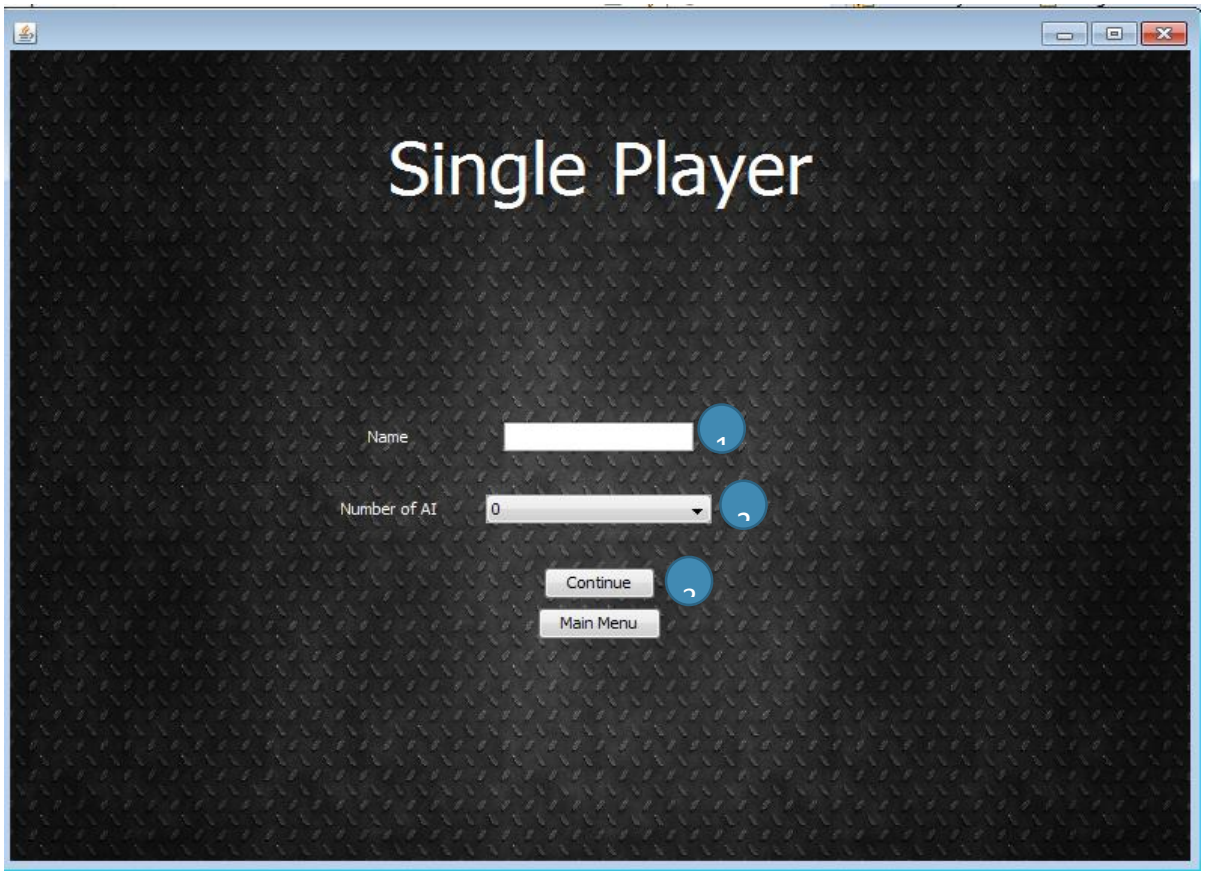
JOINING MULTIPLAYER GAME.



Join Game Panel Elements	Description	Usage
1. Player Name 	Player name. This will be the name of the host in the lobby.	Type your name into the text field and try to keep it PG.
2. IP address 	This is the only piece of information you need to get from the host. In order to connect to him.	Make sure that your firewall has an acceptance rule for java programs then enter in the IP address that the host has on his game.
3. Port Number 	This is the port in the computer that the pc will use in order to interface with	This is set by default to the value of 53555 which is a port for java but can be

	the game server.	changed by advanced users but selecting the change option to the right of the box.

SINGLE PLAYER GAME.



Single Player Panel Elements	Description	Usage
1. Player Name 	Player name. This will be the name of the host in the lobby.	Type your name into the text field and try to keep it PG.
2. Number of AI 	This is where you can choose the number of AI you want in your game. This can be up to 3 AI players making a max of 4 including the player.	Select the drop down box and choose from 0,1,2,3 AIs to add to the game lobby.
3. Continue 	Once all of your settings are as you want them you can select this button to proceed to the next step of the game setup	Once continue is selected the game will take you to the lobby screen where you can change the teams of your AI player, select a map and change any other settings you desire.