Project Report

Team C5

The Four Kingdoms of Kralpridan

Jordan Bell
Alexander Coleman
Priyan Mistry
Krishna Patel

1. Contents

2. Introduction

The aim of this project was to implement a game has that has a competitive and cooperative aspect. These aspects are quite general, allowing a lot of control over the type of game we would make to fulfil the task. Various software engineering techniques and teamwork skills were used to develop the game.

The Four Kingdoms of Kralpridan, henceforth referred to as FKK, is an RPG adventure game with a turn based battle system. FKK supports up to four players in networked play. The aim of the game is for a team to explore the world, battle A.I. and collect items to increase the individual player's power. At the start, a time limit is chosen. After the timer reaches zero, the teams are taken into a multiplayer battle to fight against each other.

The game world is split into four zones, each with increasing difficulties. Players can walk on the grass terrain to have a chance of entering random monster battle. Treasure chests are placed around the map containing randomly generated items of various rarities. There are also non-playable characters placed around the world, who give quests and reward items upon completion.

The cooperative aspect of the game is key to winning the multiplayer battle. Teams of player are able to communicate using a networked chat system, and may coordinate their plans. Players can organise into roles via initial character selection. Users are also given the option to build their own roles by learning skills in the available skill trees. For example, one team member may build offensively, or another may specialise as a healer. Players in the same team can also trade items through the network, sharing their wealth to further empower the team. In the final multiplayer battle, teams will need to work together in order to defeat the enemy. It will be important to communicate tactics via the chat system in order to win.

The battle system supports battles between player controlled characters, auto-players and AI-controlled Monsters.

This report will cover as wide range of topics, including FKK's game design, software design and class structure, as well as how the game's function as a whole. The team's roles and management techniques will be outlined in detail.

3. Software Design

3.1 Overview

The game was built from a number of main modules. This included the explorable world (hereby referred to as the Overworld), the battle system, the networking and the UI Framework. The Game Engine acted as a root system upon which everything else was built, and so will be mentioned here alongside the main modules. The UI Framework has been described in more detail in the GUI section of this document.

Modular design was a large part of the design process, acting as a Top-Down foundation for the rest of the subsystems. This allowed the team to easily decouple independent aspects of the software and develop them separately until integration began.

UML Class diagrams for these systems can be found in the appendices.

3.2 Modules

3.2.1 Engine

The software starts at, and is managed by, a Game State Manager. The program is built around Game States, ie the Main Menu, the Overworld state and the Battle state. Each Game State has a set of data to be updated and a number of graphical elements to be rendered. The data and images are encompassed in a class known as a GameObject: Any object within the game which requires a constant call to update and render.

3.2.1.2 Game States

The Game State Manager was designed as a globally accessible singleton automaton. From anywhere within the program, the manager can be accessed to switch game states or end the program. Additionally, the Game State Manager handles initialisation and termination of the auxiliary libraries before and after the main game loop.

Once running, the Game State Manager calls upon an update and rendering hierarchy via the currently selected Game State. This Game State has a list of Game Objects whose update and render functions are called. Ordinarily, the list of Game Objects within the game state will be small. Not all active Game Objects are nested directly within their parent Game State.

3.2.1.3 Game Objects

Each Game Object has a dynamic list of children: other Game Objects that are nested within a parent Game Object. This is a two-way tree structure, and contains a plethora of convenient functions, including, but not limited to, parent-relative movement, absolute positioning and child search functions.

When a Game Object's update or render functions are called, every child Game Object's corresponding functions are also called. This allows a number of useful Game Object relations, such as a moveable UI Window containing several UI elements, each sub-classing the Game Object class. The recursive structure and function calls proved suitably efficient, allowing the game to support tens of thousands of separate game objects within a single game state (see section 3.2.2).

3.2.1.4 Images

Additionally, every Game Object has:

- An Image: The graphical component of the object. This is an abstract class, whose three implementations are subsequently listed below.
- An optional clip: A rectangular area specifying what portion of the image to render. If left null, the entire image is rendered.

There are three different image implementations:

- Texture Image: A conventional image loaded from an image file. This implementation wraps around an OpenGL Texture, which is used by the SlickUtil library.
- Rect Image: A primitive rectangle shape. Can be set as a filled rectangle or as a rectangular outline with a set line size. A good example of this image would be in the loading bar at the beginning of the program. The RGB colour of the rectangle can be set.
- Font Image: This is an image created from a given text and font. The font can be a native font, or one loaded directly from a True Type Font file. The image is essentially a wrapper class around the True Type Font class's drawString function.

By making the abstract Image class, we were able to have all of the benefits of an interface (i.e. abstract rendering and size formatting) with the addition of universal image functionality, such as the ability to render an outline around the image, used in GUI testing.

3.2.2 The Overworld

Overworld design began at the start of the development period. Before the exact dimensions and layout of the world was designed, an algorithm was written to create a unique (random) grass pattern generator. Given a rectangular area, the algorithm would create in-game grass objects and add them to the world Game Object. Everything within the world is added to a World object - this way, moving the camera became a simple matter of moving the world. Everything within it (all but the game's GUI) would move along with the world game object via relative motion (section 3.2.1.3).

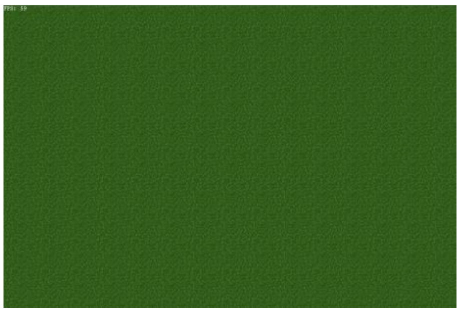


Figure 1: The algorithm starts fresh, placing basic grass tiles at the start.

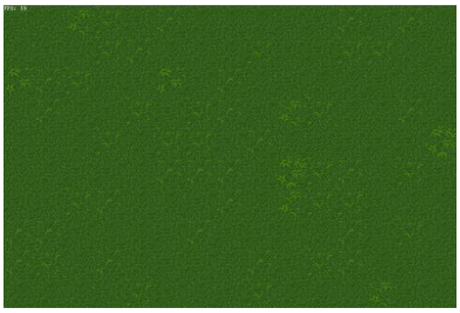


Figure 2: The algorithm's second run sets random grass thicknesses at random locations.

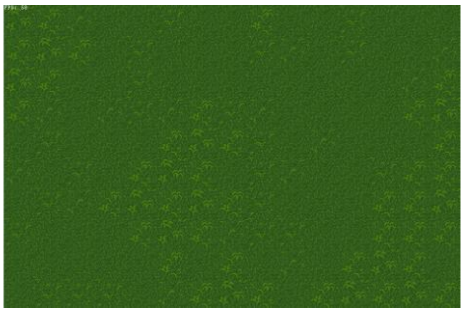


Figure 3: The algorithm's final stage expands each thickness outward, to create a smooth transition from thick to thin.

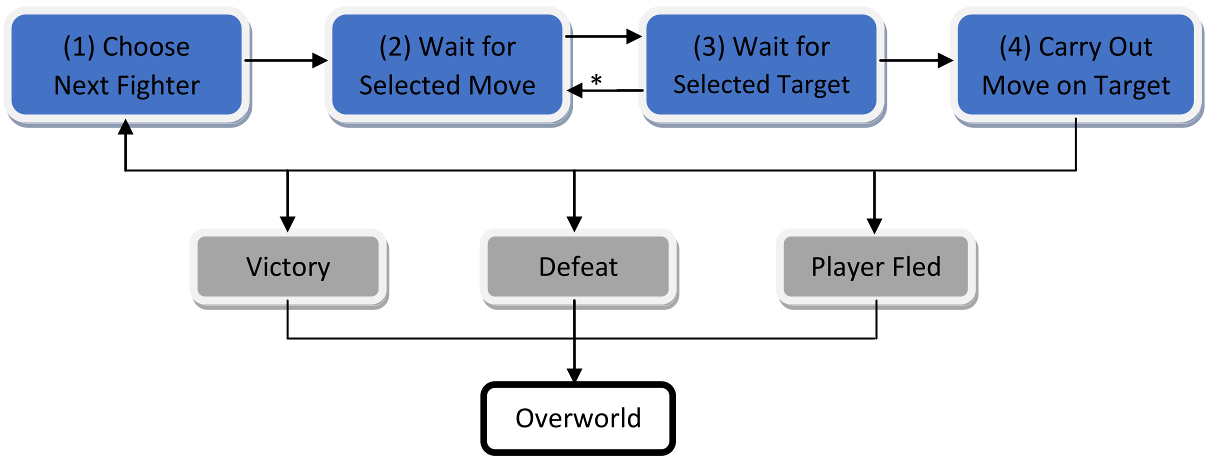
The world itself is a Game Object, split into a number of 64 by 64 pixel tiles. Each tile is cut from an environment tile set (loaded image), and uses the Game Object's image and clip support to render a unique portion of that tile set. Each tile (hereby referred to as World Object) is assigned a set of properties. This allowed other members of the team to design the World Objects freely. Each World Object instance may or may not collide with the player, may or may not span across several grid positions, may or may not travel (change positions at runtime), and may or may not be interacted with. A World Object's on Interact function is called when an adjacent player presses the interaction key [F]. For example, for treasure chests, the class's interaction function was overridden to generate and reveal a number of items for the user to collect.

After the world was designed on paper, the process for building it began. To simplify the process, various tools were written to instantiate world objects. For example, given a rectangular area, lakes could be built using a water generator. Pathway algorithms were also written, along with various flora generators. These tools proved invaluable for quickly and easily creating the world.

3.2.3 The Battle

Each battle game state is started anew. Unlike the Overworld, the Battle State was designed to be different every time it began, reinitialising with new fighters and new user skills.

The battle system in itself, operated on a state system. Similar to game state switching, battle states would transition from one to another as follows:


(1) Choose Next Fighter
(2) Wait for Selected Move
*
(3) Wait for Selected Target
(4) Carry Out Move on Target
Victory
Defeat
Player Fled
Overworld

**Players can revert their move decision during target selection, returning to the move selection state.*

Operating with different Battle States allowed the game to behave differently at different stages. The Game State's update function would fork (via a switch statement), based on the current battle state, calling separate update behaviours for each of the battle states. For some states, such as Choose Next Fighter and Carry Out Move on Target, this usually involves a single update call before proceeding.

For Battle States in which a loop waited for a human or AI decision, the system would simply do nothing until a decision was made. The Battle Game State contained a pointer variable to a selected move, and a selected target. If the current fighter was an AI, its answer would be received immediately, and the game would advance. However, for user input, a decision wouldn't be made until the corresponding input buttons were clicked. Decision variables could be publicly set, thus creating a safe algorithm which would fail to advance until the system was sure of a selection.

3.2.3.2 Fighter Order

The first battle state would analyse the battle's fighters, and determine the next fighter to have a move. A fighter's speed stat would directly affect its order, however it was not as simple as ordering the fighters based on their speed. The fighter class implements Comparable and compares objects based on a readiness variable (If two fighters' readiness values were equal, the one with the greater unique ID number would take precedence).

Each round, a fighter's readiness is incremented by their speed value. When a fighter has at least 100 readiness, they are considered to be ready to fight. The fighter with the greatest readiness is given control, and is prompted for a move and target decision. Once a fighter has made a move, their readiness is reduced by 100, and the next fighter is selected.

This system was used to create a subtle sense of speed. If a fighter with 60 speed fought against a fighter with 40 speed, the quicker fighter would attack three times for every two attacks from their opponent.

3.2.3.3 Moves

Each move was defined in a separate file, and accessed via a public enumeration. When a move was selected by a fighter in the battle state (either by user input, or by AI choice) the system would notify all players of the choice, the user, and the target via a GUI notification system. This move's effect was then evoked in the final state.

The move enumeration was designed so that each value had a dynamic onEffect function. This function was passed to the object using Java's new lambda function support. The lambda functions provided simple, in-line definitions of the move's effect. It was then a simple matter of a selected move's onEffect function being called, and the game would handle the rest.

To work with HCI, each move's function would return a string. This string would be used as a message to all players. For example, a move which dealt 10 damage to a target, would return a string such as:

"Target received 10 damage."

Some moves had multiple effects, and so could separate lines using the newline escape character supported by the UI framework. For example, a move dealing 10 damage to a target and healing the user for 10 points would return the message:

"Target received 10 damage\nUser was healed by 10 points."

The two parts, separated by the new line character, would be parsed into separate lines by the UI system.

4. Game Design and HCI

4.1 Overview

The features and design of the game are some of the most important aspects to consider when creating the game. FKK was inspired by various popular games such as Pokémon(Game Freak) and Final Fantasy(Square Enix). However the game's take on the RPG genre demonstrates how FKK stands out from other games with mechanics and designs not found in more popular games. The following game design decisions were made in order to create a unique cooperative and competitive experience.

4.2 World Design

The game's world was designed using various image objects, and has been populated with various types of trees, grass, paths and water tiles. The map is split into 4 zones, each with increasing difficulty. These have been named Bell Town, Letap Lands, Coleman Cove and Mistry Valley respectively. Each zone has a unique shade which gets darker and angrier as AI difficulty increases. In addition to this, the monsters that the player encounters in each area are unique, and appear increasingly menacing to further convey the increased difficulty. This ranges from the simplest bunny to the most difficult demon.

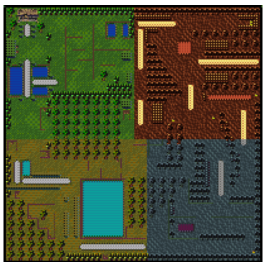
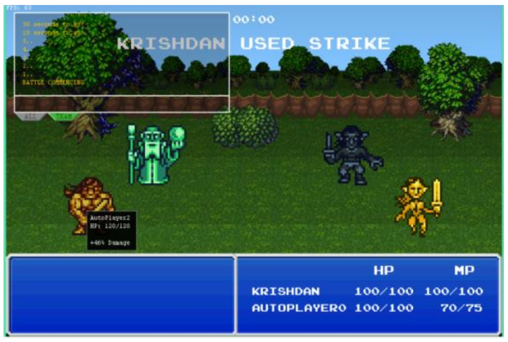


Figure 1: FKK's procedurally built Minimap, showing all four playable areas.

4.3 Gameplay Mechanics

4.3.1 Battle

In an A.I. or multiplayer battle, every entity has a turn to perform a move. The order of move is defined by the speed statistic which every player has. The higher speed a player has, the more opportunities there are to attack (section 3.2.3.2). Players can choose from a selection of attacks which can either attack the opponent or buff themselves or their teammate. A player will do increased damage if they have more attack statistic than the targets defence statistic.

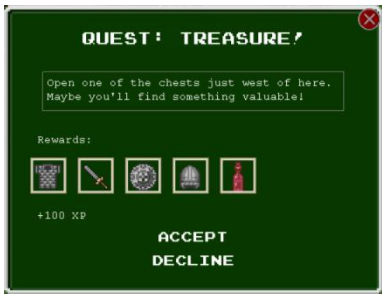


4.3.2 Pets

In the local portion of the game, the player starts with a bunny pet. However, during battles, players have the ability to ‘tame’ monsters to capture new pets. Monsters can be used in battles to fight alongside players. A player can only have one pet following them at any given time, and previous pets are left behind in the game world. Players have the ability to revisit old pets, swapping them out at will.

4.3.3 Quests

There are non-playable characters in the world which distribute quests. In return for completing quests the player is rewarded with experience and items. Quests can vary for example there is a quest to find an item chest and open it, or kill 5 monsters in a particular zone.



QUEST: TREASURE!

Open one of the chests just west of here. Maybe you'll find something valuable!

Rewards:

+100 XP

Figure 2: A quest popup window.

4.3.4 Items and Trading

Players can collect items from chests and by winning A.I. battles. These items consist of power-ups which can replenish or boost statistics, and gear which occupy player equipment slots on the head, chest and arms. Gear items have statistics which are dependent on the rarity of the item. Rarity is conveyed by various colours surrounding the items image. These range from grey (common) to orange (legendary). A legendary item will have less chance of appearing in the game than a common item. During the game, players are able to trade items over the network to team members.

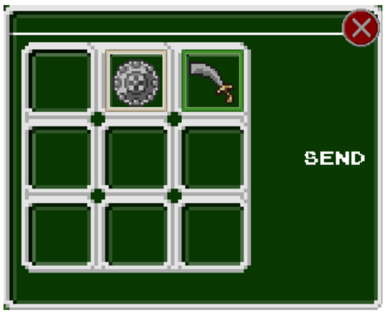


Figure 3: The item trading interface. Items are dragged into the cells.

4.3.5 Skill Trees

Players have the ability to choose skills to differentiate themselves in the game and fit into different roles. There are three separate skill trees: Offense, Defence and Support. A player gains a skill point for every level they gain. Skill points can be spent on skills in the tree hierarchy, giving player's a specialisation choice. Some abilities have prerequisites from two different trees, allowing players to choice of hybrid specializations. For example, a player can go down the offence and support tree to learn the Drain Health skill, which has the ability to damage the target and heal the user at the same time.

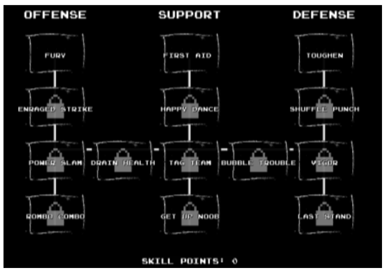
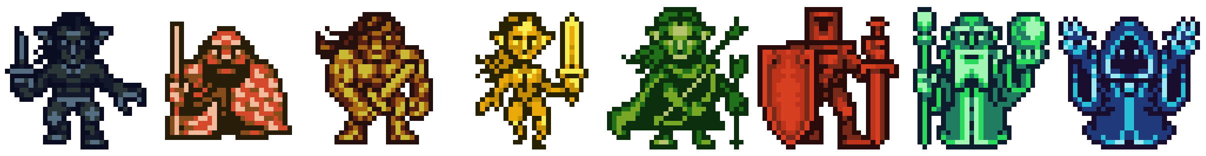


Figure 4: The skill tree menu. Locked skills are made available when above skills are unlocked.

4.3.6 Character Selection

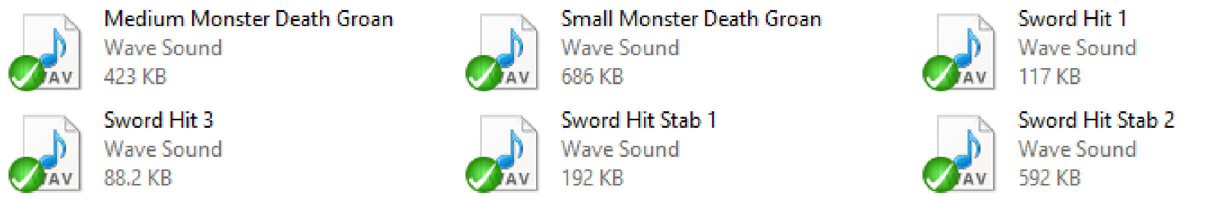


Users can select characters at the beginning of the game to further differentiate themselves. There is a selection of eight characters, each with different Attack, Defence, Speed, HP and MP values. If a user wishes to take an offensive role, they may select a character with higher attack at the cost of their defence value.

4.3.7 Boss Battle

In the final zone of the game there is an altar which, when interacted with, initiates a boss battle. This is an end game feature, aimed at players who have successfully reached high levels, maximized their gear and acquired a strong collection of skills. The final boss can only be accessed through a high-level quest found at the entrance of the most difficult zone, and rewards the player with a substantial stat boost via equipment and gained experience.

4.3.8 Sound Effects and Soundtrack

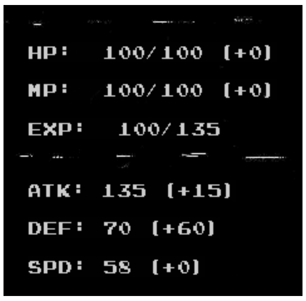


Medium Monster Death Groan Wave Sound 423 KB	Small Monster Death Groan Wave Sound 686 KB	Sword Hit 1 Wave Sound 117 KB
Sword Hit 3 Wave Sound 88.2 KB	Sword Hit Stab 1 Wave Sound 192 KB	Sword Hit Stab 2 Wave Sound 592 KB

The game has full audio, complete with separate volume toggles for the music and sound effects. The main menu has its own audio track: a pleasant piece which sets an introductory feeling to the game. Once a match begins, a more upbeat, fantasy track is played while the player explores the world. Entering a battle can trigger various music tracks. Additionally there are battle sound effects. Sound effects are paired with, and played by, different fighters when hit. Additionally, moves let off a variety of battle sound effects.

4.3.9 Fighter Statistics

A player's statistics are comprised of health, magic, attack, defence and speed.



```
HP:   100/100  (+0)
MP:   100/100  (+0)
EXP:   100/135

ATK:  135  (+15)
DEF:  70  (+60)
SPD:  58  (+0)
```

- **Health:** Once a players or an A.I. entity has their health completely depleted they will be die, removing them from the battle.
- **Magic:** A player can spend magic points to use their abilities. Once their magic is depleted they can only use basic moves such as strike at no cost.
- **Attack:** Attack will increase a players damage potential. If a player's attack is higher than the opponent's defence they will do more damage.
- **Defence:** Defence will decrease incoming damage, if a player's defence is higher than the opponent's attack, they will receive less damage.
- **Speed:** Speed will increase a fighter's player attack frequency. The higher their speed statistic the more opportunity they will have to attack (see section 3.2.3.2).

4.4 HCI

4.4.1 Overview

Given the genre, HCI was a very important focus. The game would have to make a large amount of information available to the user, and users could be easily confused if this wasn't done correctly.

4.4.2 Lo-Fi / Hi-Fi Prototyping

The team's HCI efforts began early in the development process, with Lo-Fi media prototypes. Paper prototypes were drawn up, starting with each stage of the main menu (see appendix I). This provided a basic idea of what the game would look like, as well as giving the team a chance to envision and tweak components of its design.

The battle and world UI prototypes were drawn on paper as well. However, their main prototype was in digital media.

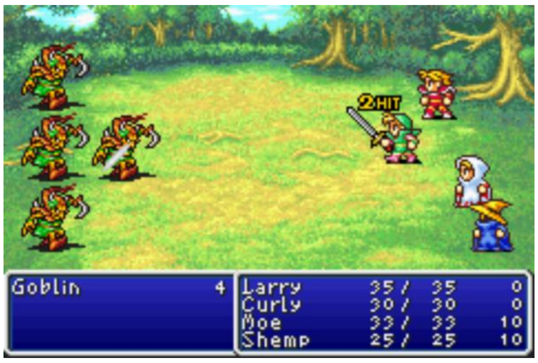


Figure 5: A Battle Scene from Final Fantasy I (Square Enix)

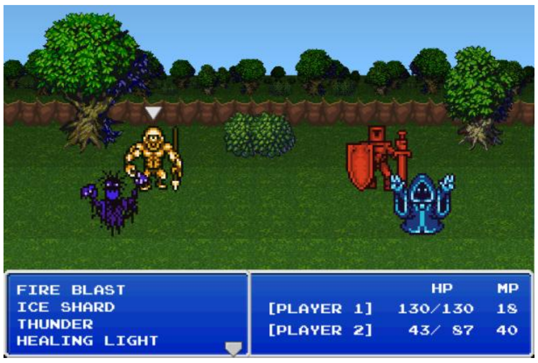


Figure 6: An early Hi-Fi screenshot of FKK's Battle Scene.

Before the game was built, the art assets were gathered and constructed, in Photoshop, into two mock screenshots, one for the world (appendix K), and one for the battle scene. This also gave a chance for users to view how the game would look before it was built. The scene itself, with its button and label placements, was modelled after existing games of the genre and was well accepted in early user tests.

4.4.3 User Response

Early on in the game's development, in-game UI elements were simply scanned drawings from the Lo-Fi prototype. This was intended as a temporary placeholder. However, during early user testing, users frequently noted on the placeholder scans, expressing positive opinions on its appearance. For this reason, in many cases, the scans were kept in the final version of the game.

During the user testing session, many users noted the ease with which they could play the game. One user wrote in their feedback form, "It was quite easy to use, the user interface was fantastic, well done." Another user agreed, saying, "Usability is well implemented." The game received an average usability of 92%.

However, specifically, users reportedly found the controls for target selection unintuitive. Additionally, in regards to the item equipment screen, one user wrote, "Unclear which arm equips what kinds of weapons in the inventory screen." To remedy this, a clearer diagram was drawn to clarify weapon slots.

5. GUI / A.I / Networking

5.1 GUI

The UI framework was built upon the engine, described in more detail earlier in the report. Every class within the UI package subclasses GameObject via a universal class, UIElement. The framework design took influence from the Swing library, however it was entirely written from scratch. The high-level project requirements for UI included:

- Input boxes for player information
- Buttons for menu navigation
- Radio buttons for chat channel switching
- Tick boxes for sound options.
- Labels
- Draggable windows

5.1.2 UIElements

The UIElement super class extends from everything in the Game Object class. Additionally, UIElements can be hidden and revealed by toggling a Boolean value. They can also be aligned relatively to any UIPanel that they are nested in.

UIPanels are conceptually similar to JPanels. Only UIPanels can have UIElements nested within them, including other UIPanels. A UIPanel is a subclass of UIElement, and thus can be hidden and revealed on command. This affects all nested UIElements within that panel. This has been used to great effect for popup windows, and "menus" within the Main Menu game state. Switching menus is a simple matter of hiding and revealing different UIPanels containing each menu's information.

For easy formatting, UIElements can be aligned (relative to their parent UI Panel) as such:

- Align Centre (Horizontally and/or Vertically)
- Align Left / Right / Bottom / Top
- Positioned some percentage, horizontally or vertically, along the length. ie Vertical, 0%, places the element at the top, while 60% places it slightly below centre.

Alignment uses the abstract Image class to good effect, utilizing each UIElement's Image size to calculate the coordinates required to make each alignment. This is built on the Cartesian coordinate system of Game Objects. Additionally, a UIElement's X and Y coordinates can be tweaked manually, just like any other Game Object.

5.1.3 Buttons

Buttons work in tandem with LWJGL's Mouse class, using mouse coordinates and a bounding button area to determine if the player has:

- Clicked on the button
- Clicked off the button
- Mouse released the button
- Hovered over the button
- Hovered off the button

Each button has a set of abstract functions which are called upon each of these events. When a new button is instantiated, these functions can be overridden with unique behaviour upon that event.

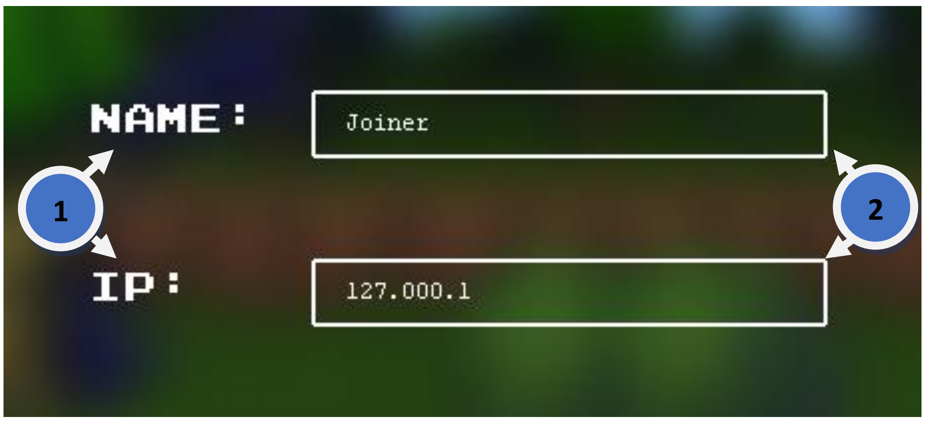
5.1.4 Draggable Elements

There are two areas of the game in which draggable elements feature. The first is when dragging items to, from and within the inventory, chest and trade menus. These are simply buttons with images, whose on-screen coordinates are changed as a mouse moves while held down. Each draggable item contains an Item object whose information can be accessed and used throughout the game, such as improving the player's stats and health.

Some windows, such as the trade menu and treasure chest interface, needed to be draggable around the screen, similar to conventional Operating System windows. This allowed the player to position the windows into convenient positions. Specifically, this allows the user to move treasure chest and trade windows around, making it easier to drag items to and from their item grids.

5.1.5 Labels

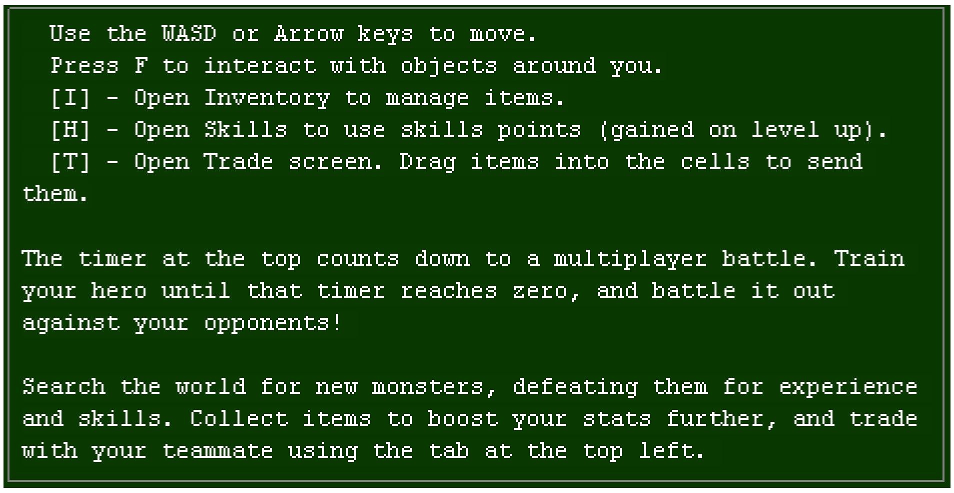
Labels are simply UI elements that contain a FontImage as its image implementation. Simply put, they are labels of text, joining together font functionality with a game object in a GUI context.



(1) Labels, displaying text on screen.
(2) Text input boxes. These contain buttons, which detect user selection, and have internal labels which are added to when a character is typed.

5.1.6 Text Boxes

Text Boxes function similarly to traditional text boxes, in the sense that they can contain and display a large amount of text, with wrapping, within a multiline area. Their width can be set, and their height variable based on the length of the text set to them. These text boxes are not as dynamic as traditional text boxes, however they work well with invariable strings.

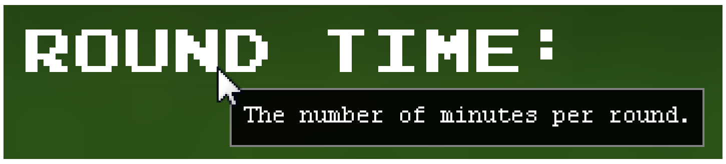


```
  Use the WASD or Arrow keys to move.
  Press F to interact with objects around you.
  [I] - Open Inventory to manage items.
  [H] - Open Skills to use skills points (gained on level up).
  [T] - Open Trade screen. Drag items into the cells to send
them.

The timer at the top counts down to a multiplayer battle. Train
your hero until that timer reaches zero, and battle it out
against your opponents!

Search the world for new monsters, defeating them for experience
and skills. Collect items to boost your stats further, and trade
with your teammate using the tab at the top left.
```

Figure 7: A textbox in a help window, explaining the game to new users.

These boxes were designed for multiline text wrapping. To do this, the box creates a dynamic list of Label objects which are displayed one after the other. This is a great example of the recursive Game Object structure, with its relative positioning, which allows the Text Box to hold several UIElements and move as if one image.

5.1.7 Tool Tips



ROUND TIME:

The number of minutes per round.

Optionally, hovering over a UIElement will reveal its Tool Tip. This nests a button, known as a ToolTipHoverComponent, within that UIElement. This button's onHover function reveals a textbox filled with information as set by the designer.

For example, hovering over an item's image within the game will reveal a box containing information about that item: its stats, name, rarity and description. Tool Tips are built into the UIElement class and can be set with a simple function. These are used heavily for HCI, providing quick and clear explanations for anything that the users might not understand. Additionally, they enhance user experience by neatly and selectively revealing relevant information, such as displaying fighter information during the battle state.

5.2 A.I.

An AI component is assigned to a fighter if it is either an Autoplayer in the final battle, or a monster encountered throughout the length of the game. An abstract superclass, AIFighterComponent, is used as the interface for several AI implementations that have been written for the game. An AI Fighter Component can be prompted for a move decision, or a target decision. Alternative versions of each function exist, which return special serialisable versions used in the networked battles.

Several AI "personalities" were created for different monsters in the game. Two personalities were written for each attack style: offensive, or defensive. They are as follows:

5.2.2 Offensive Personalities:

5.2.2.1 Aggressive AI

- Only uses offensive attacks
- Motivation: Killing opponents as fast as possible.
- Targets the opponent with the lowest health.

5.2.2.2 Bully AI

- Only uses offensive attacks
- Motivation: Inflicting the most damage against opponents.
- Targets the opponent with the lowest defence.

5.2.3 Defensive Personalities:

5.2.3.1 Defensive AI

- Uses a mix of offensive and defensive skills
- Motivation 1: Aid the most defenceless ally (including self)
- Motivation 2: Attack the most defended opponent
- Targets the opponent with the highest defence, or the ally with the lowest defence.

5.2.3.2 Communist AI

- Uses a mix of offensive and defensive skills
- Motivation 1: Take down the biggest opponent, targeting the healthiest one.
- Motivation 2: Keep allies far from death, and empower them to fight.
- Targets the opponent with the highest health, or the ally with the lowest health.

Each of the above traits determine the AI component's choice when prompted in battle. If a fighter's AI component is non-null, it is prompted for a response. However, during battle, if no AI component has been assigned to a fighter, it is assumed that it is user controlled, and will wait for the user's input.

All data related to a particular monster would carry over between battles, in the case that that particular monster was tamed by the player. A tamed monster would become the player's pet, whose fighter model carried over between battle states. This pet then follows the player from position to position on the world's grid layout, and can be retrieved at any time if switched out for another. Thus, any monster's assigned AI will remain throughout its life.

5.3 The Network

5.3.1 Overview

We chose to implement a User Datagram Protocol (UDP) Network. UDP was selected over a Transmission Control Protocol (TCP) because of its speed advantage and also UDP is simpler to set up for multiple clients. We also elected to do the majority of the calculations on the local computers leaving the server to pass data to the player's computers and only handling essential calculations. The majority of the packets sent are serialised objects. This was done because with objects it is easy to send multiple pieces of data in one packet without the need for string manipulation and when a client receives an object it can be passed straight into the game engine.

5.3.2 Server

The server side of the network is comprised of one main class (networking.server.Server.java) and several helper classes (the other classes in networking.server). When a packet is received by the server it tries to convert the byte[] to a string. If this is successful and the string starts with "//" then the server knows the packet was sent as a string and sends it to the processString method. Otherwise it deserialises the byte[] and checks what type of object it is. It then calls the relevant process method. For the majority of packets the server processes them, updates its clients lists and sends them on to the clients for them to do the same.

The main functions of the server other than passing packets to players is maintaining the game clock, managing client connections and dealing with the AI in the multiplayer battle when the clock runs out. The game clock runs on its own thread and simply checks the systems time (in nanoseconds) when the game is started, adds the game length in nanoseconds onto the start time and checks when the current time matches this end time. When it does this the server prompts all clients to send their Fighter Information to the server. The server passes this information to all the clients and waits until it has received as many Fighter Information's as it has clients. Then if it doesn't have four it generates AI fighters.

The AI the server generates has stats within *±25%* of the average stats of the human players in the game. This prevents there being seriously over or under powered AI player and tries to ensure the game is enjoyable.

When a player quits the game their client sends a notification to the server but the server had to handle clients disconnecting without sending the leaving notification. To do this we wrote a class called ManageClients. This class sends each client the "testConnection" string: "//cpc/a/e/". The client responds with its unique playerID which is added to a response list on the server. Then after four seconds the server checks which playerIDs are in the response list and clears the list. If a client hasn't replied after this four seconds the server adds one to that clients response count and if the new response count is 5 the server will kick the client out. The server then re-sends the "testConnection" string and the process starts over.

When it is time for an AI to make a move in a multiplayer battle the host requests a move from the server. The server makes its choice by calling the same function that is used in single player AI battles. The move choice is then passed to all the clients, at which point the host requests the AI target. Again the server makes its decision using the same methods that are used in the single player battle. The server then passes this target to all the clients and the game progresses.

The AI move and target selection also employs a similar response system to the one used to ManageClients class to make sure all the clients receive the move and target and that the game doesn't move on until everyone has received the move and target.

5.3.3 Client

The client is far simpler than the server because it doesn't need to handle the game clock or AI. It has two main function:

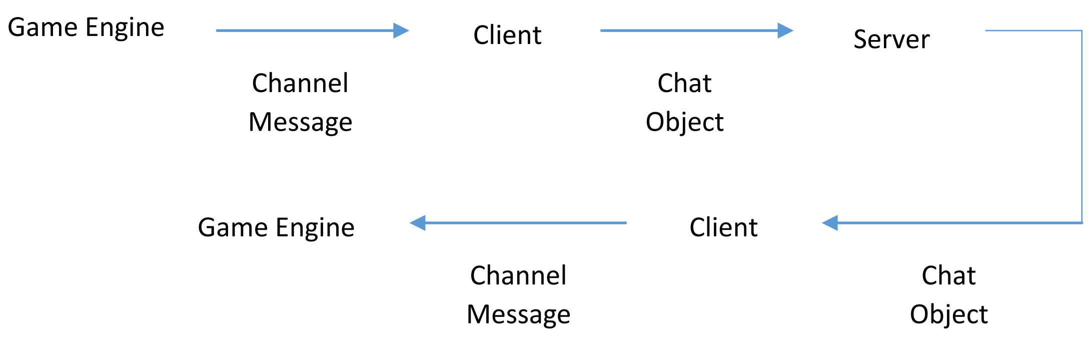
1. Construct and serialise objects from the game engine and send them to the server
2. Receive packets from the server, deserialise them and pass them into the game engine.

All the processing the client needs to do is update its list of other players in the game. Each client needs to maintain an up to date list of all the other clients so they know about any level changes or changes to any other stat. In the final version of game we actually didn't use the stats the network holds other than player level, but we decided to keep the ability to store player stats on the network. We kept this functionality so in a future version of the game the server could monitor player stats in an effort to prevent players from cheating.

5.3.4 Network Objects

Each network object is a serialisable object that has the sole purpose of transferring data over the network and provides getters and setters to get the variables out of the object so they can be used by the recipients. An example of a network object is the ChatObject.java class.

The Chat Object is used to send chat messages across the network. The chat object contains a string which is the message, the senders team and an enumeration called Channel, which tells the recipient if the message should be displayed as a team message, public message or notification. There are no setters for this object as when you create the object you pass it all the variables but there is a getChannel(), getTeam() and getMessage().



An example of data transfer over the network. In this diagram a chat message is being sent from one player to the rest.

6. Software Engineering and Risk Management

6.1 Overview

Due to the nature of the project it was evident that we would need to employ good software engineering practices in order to complete the project to the highest standard and on time. We had to take into account several factors such as the time period of 11 weeks, the amount of coding, producing a prototype and the documentation required for the project. As a result we decided to use an iterative development model with prototyping approach to complete our project.

Using an iterative model allowed the team to carefully plan out each stage to create the game as we were aware from the assignment brief or the requirements the game already needed. It gave us the opportunity to separate the stages across the 11 weeks but also add new features based on feedback from tutors, peers and during our prototype presentation.

6.2 Software Engineering Techniques

6.2.1 Requirements Definition

For our project we were given a task to implement an online game that has cooperative and competitive aspects. For this project we have several features the game was required to have which included

- A number of competing teams
- A team of cooperating players
- Different player roles
- An A.I. component
- A random element
- A Graphical User Interface
- The game must be playable over the network

In addition to these game requirements the overall project had several criteria which included

- Delivering the software on time
- Keeping resources within budget
- Deliver software that meets requirements
- Maintain a functioning team.

Before the next stage of our project we created a Software Requirement Specification document to outline the various requirements for the system.

6.2.2 Planning

We planned the project using a Gantt chart separating the tasks in respect to the 11 weeks we had to complete the project and the prototype presentation that was required in week 6. In respect to our development model we planned to design and implement the game in a span of 40 days and be ready to deploy our first iteration of our game for the prototype presentation and evaluate the from the feedback that was given for further iterations of the game.

6.2.3 Iteration 1 - Prototype

6.2.3.1 Design

We designed the core elements of the game which included the:

- Battle system
- Random monster spawns
- Multiplayer networking
- Overworld elements
- A.I. component
- User interface

6.2.3.2 Implementation/ Testing

The aim of this iteration was to complete the basic functionalities the game required and demonstrate it. We coded as a team to implement our designs and as a result had a basic game with a client that multiple players could connect to, a graphical user interface, a basic battle system, A.I and a main game world where the player can run around.

The game engine was modular so we could keep the game integrated as much as possible at one time, we did a large amount of integration testing before the prototype presentation to ensure we presented one client of the game with all the features rather than different parts of the game.

6.2.3.3 Deploy

We demonstrated our prototype at week 6 and received feedback accordingly for the game.

6.2.3.4 Evaluate

The team took aboard the feedback we received and proceeded to add and improve features in the next iteration of our game. The feedback told the team what was required:

- An improved A.I.
- A more stable network
- A fleshed out battle system
- A larger focus on testing (We had planned this in advance to do this at the end)

6.2.4 Iteration 2 – Public User Testing

6.2.4.1 Design

We improved the game based on the feedback given at our prototype presentation and also designed new features to implement for our public testing session which included the following:

- Skill trees (Offense / Defense / Support)
- Pets
- Items, Inventory, Reward
- Player gear slots.
- Game zones and mini map
- Player levels and experience.
- Networked battle
- Item trades over the network

6.2.4.2 Implementation / Testing

The aim of this iteration was to add features to the game to present it to our peers and for them to able to play the game and give us feedback as a result in the form of user testing. We successfully implemented the features required for a full run of the game from beginning to end.

6.2.4.3 Deploy

We deployed the game for public user testing at the end of week 7 to allow other teams to play our game. Following this each person who played completed a questionnaire giving us feedback on their experiences with the game.

6.2.4.4 Evaluate

After the user playing session we found that there were several areas of the game that needed to improve upon which included

- Increased A.I. difficulty
- More help options
- Less A.I. battles and more ability to explore the world.
- Better monster targeting options

6.2.5 Iteration 3 – Final Submission

6.2.5.1 Design

At this point in the project we had completed the main features of the game, we further balanced the game and improved the game based on player feedback. We had the opportunity to add extra features to the game that we hadn't originally had in the scope. These included

- Quest system

- Adding story to our game
- Character selection and character stats
- A final boss battle

6.2.5.2 Implementation / Testing

By the second iteration we had completed the game according to our initial requirements and planning. So this third implementation gave us the opportunity to be more creative with the game adding elements of our own choosing and those which fit in an RPG game. It also was in preparation for the final deployment of the game to present it at the final presentation.

6.2.5.3 Deploy

We successfully showed our game at the final presentation in front of some of our peers and a final board of people who were looking to buy our company and game.

6.2.5.4 Further iterations

If our company and game is bought by the company we aim to continue further iterations of the game in order to develop it users with many more functionality, features and content.

6.2.6 Risk Management

6.2.6.1 Identifying Risk

For a project to be successful risk management should be undertaken throughout the duration of the project. Risk management is key when there are unexpected developments or problems in the project, it is key to have procedures in place to manage the risk effectively and have minimal impact on the progress of the project.

In order to create a good risk management strategy it is important to identify the possible risks and set out a strategy to tackle then in a priority order. Additionally it is important look at the project itself and look at the how the different parts of the project are connected and decide which have the most importance over others as they create the most risk.

6.2.6.2 Managing Risk

We aimed to tackle our risk in a prior order as follows:

1. High probability – High Impact Risk.
2. High Impact – Low Probability Risk.
3. Low Impact – High Probability Risk.
4. Low Impact - Low Probability Risk.

We identified the potential risks that could have an effect on our project and outlined a strategy for each type of risk (Our risk strategy plan can be seen in more detail in appendix J)

7. Evaluation

This project is evaluated against the SRS, Software System Requirements. The SRS document defines all the requirements and features that are going to be implemented for the game we have created.

From the SRS document it shows we have stated that the game we intended to create is an RPG based game which included features such as levelling, battles and an environment. In section 2 of the SRS, the 'Product Functions' states what functions our game must have. This included playing against AI players, teams with combination of player and AI, randomly occurring monsters. From what we have achieved in the end product, all of the functions of the product stated in the SRS document have all been completed but one, which was implementing the rounds for the game.

We planned our final game to be an easily understandable game for users with game experience ranging from intermediate to advanced – where the users are of a relatively young age between 15 – 35 years of age. We paid a lot of attention to adding H.C.I features such a tool tips and a tutorial page. As part of our testing we chose to do User testing which included giving our game to users which met our target audience. From here we received quality feedback on the HCI on how it could be improved, for example selecting a target in the battle game state and making it more clear on how to carry out a move. After the user testing we were able to gage on what we needed to change in order to make the game more usable and made the changes accordingly.

All of the Functional requirements have been met. Whilst playing the game you are able to see a 2D map on-screen which is a 100X100 grid of 64x64 pixel blocks – this map also has collisions included so you can't travel out of bounds. On this map you are able to navigate around using the direction buttons on the keyboard, the character you are navigating around the map is able to be selected prior to entering the local game. When navigating around the map you encounter battles set at 4% per tile, when this is activated you are battling an AI Monster which has its own personalities. Finally, a timer has been implemented, when the timer is up the all players go into the final team battle against each other.

All of the non-Functional requirements have been met. The performance of the game is very responsive and sends information to all clients from a server efficiently, the game runs consistently at 60 fps. The overall game is very reliable and does not disrupt the user's experience of the game. Playing single player means you do not need to connect to the network making the game available at any time. As for the maintainability of the game all the majority of the code has been equipped with full Javadoc so that other users are able to gain an understanding of the written code.

In addition to the set requirements we set the beginning of this project we were able to add a lot more functionality to make the game more usable. These features include pets (an AI which battles with the player), chests (to collect items), quests (to complete to gain rewards and experience) and a trading feature between two team mates.

In conclusion, we have met the healthy majority of the requirements set in the SRS document at the beginning of the project and added extra functionality to the game without hindering the initial requirements. So from the requirements set and the requirements that have been met, this project has been fully accomplished.

8. Team Work

8.1 Organisation

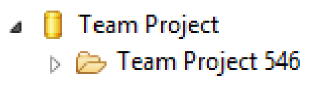
Due to the nature and size of the project it was essential for each team member to maintain good communication and be organised. As a team we constantly communicated through various means to ensure everyone had the same understanding of what was going on in the project. We used the Computer Science common room as a regular meeting point, off campus we communicated primarily through Facebook Messenger.

Prior to writing the game code we met up and decided what kind of game we wanted to create. This involved openly discussing ideas in the common room and writing potential ideas on a white board. We found we were more productive doing most of the programming as a group, we regularly sat in the common room programming together.

Day	Team Meeting	Hours
Monday	11am-2pm	3
Tuesday	10am-4pm	6
Wednesday	9am – 4pm	7
Thursday	1pm - 3pm	2
Friday	9am – 2pm	5

These hours ensured we were working at least the 20 hours a week required for the project and meant we did not have to work extremely late night at the very last minute, ensuring our pace was consistent throughout the project. We also had to work in slots where everyone didn't have a lecture so as a result these hours were preferable. Programming together ensured we all knew which team member was working on certain parts of the project. It allowed us to ask question if we were finding difficulty and discuss various ideas for the project.

8.2 SVN


Team Project
Team Project 546

Despite SVN being mandatory for the assignment, it was heartily accepted as an essential tool for the project, keeping all members of the team up to date with the latest features and updates. Commits increased during integration testing due to small changes required for testing the game's networked components. A well organized package structure and separated the game's modules and classes and ensured a minimum number of conflicts during development. Typically, team members would work well concurrently, using timely communication to avoid clashes.

8.3 Problems

There were some problems we came across during development. There were some small issues with team members who had other commitments not related to the project. To resolve this we ensured we clearly set out time when the whole team was free to meet up and work together. In some cases team members were not able to work in person and we communicated online whilst continuing to work.

9. Summary

To summarise, this final project report goes over all the different aspects of our game, The Four Kingdoms of Kralpridan, which has been explained in full detail for the 4 main areas of the project which are Software Design, Game Design & HCI, GUI/AI/Networking and Software Engineering and Risk Management.

9.1.2 Software Design

The Software Design section is a fundamental part of the project and this document as it explains the modular software design which consists of Game Engine, UI Framework, Overworld, Battle and the Networking. For each of these modules it goes into great detail of which elements are a part of which module and how these elements are make up the module and allow each separate module to come together. The software design section also covers the class structure which shows which classes and how many classes make up each different module.

The Software Design was a very important part of the project as it was the base where we could build up modules separately and make it easy for integration.

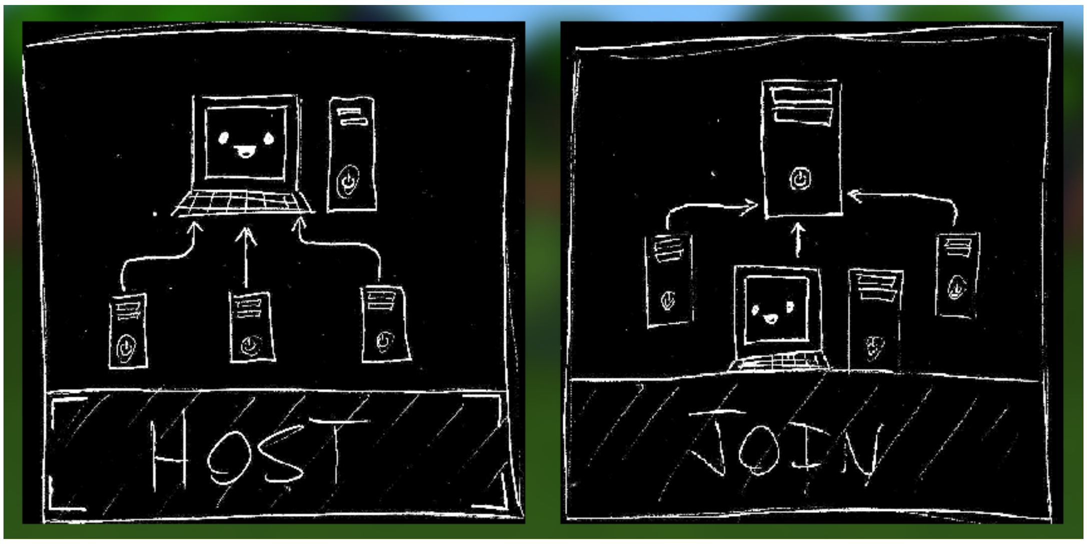
9.1.3 Game Design & HCI

This section goes into the overall game design in great detail. As it mentions the World design and how the map is going to be created, which is using image objects such as trees, grass, paths and water. This also includes how the map is laid out with having 4 different zones which increasing level of AI Monster difficulty.

This section also include the gameplay mechanics in terms of game design, this includes components such as battle, pets, quests, item and skill trees. These features are all explained in great detail at how they affect the overall gameplay of the game. Lastly, this section also has the HCI which indicates how we made our game easily understandable for the user by including every component of the UI and how the user interacts with them –this also includes feature we included to make the game more usable with features such as tool-tips and an in-game tutorial.

9.1.4 GUI / A.I. / Networking

This section has three fundamental parts of the project which are the GUI, AI and the Networking. These are very key for the game and in this section it demonstrates the features that have been implemented for each of the 3 components. For each of the components it shows how we went about completing it, for example for GUI it explains the GUI features which consists of UIElements, DraggableElements, Labels, Buttons, Text Boxes and Tool Tips.



For the AI, this section describes the AI with different “personalities” have been implemented for the Monsters, it goes into detail of how each different AI behaves when in a battle. The Networking section goes through which protocols and methods we chose to use for the game for players to connect and player over a network.

9.1.5 Software Engineering and Risk Management

This final section covers the Software Engineering techniques used and how we used the selected practice. It describes the requirements definition, planning, implementation and testing phases of the project. It also explains how we went about implementing these techniques. As for the Risk Management, this section states why identifying risks should be taken seriously. How we went about managing risk is also demonstrated in this section by stating the potential risk and the strategy to tackle the given risk.

10. Individual Reflections

10.1 Priyan Mistry

10.1.1 Overview

My experience of this group project working on a game of our own creation has been a great learning experience as we were working from scratch. As we didn't particularly know each other we first had to introduce each other and get to know how we all worked. Starting early on we got know each other we were able to start completing the work, usually spending large amounts of time in Computer Science building working together every week. So from this experience I believe getting to know each other well and how we worked well as a team was vital, as it's been a positive experience where we produced a quality end product.

10.1.2 Working in a team

Working in this great team to complete this group coding project was new for me, which was coding and creating a product in a short time period. I have learnt a great deal from this project and my team mates. As I am a relatively new to programming I was taught many new techniques in making the code more efficient by using enumerations and also making the code look presentable and of better quality. Not only this, but being able to implement different software engineering techniques such as the model used and testing phases.

At the early stages of the project we came together and established straight away that this was going to be a tough project. So our plan was to choose a simple idea, one which we could complete in the time given with time to spare. This is because we could then add features as we liked, as we had time to do this.

Coming up to the prototype presentation we met up 3-4 times a week to have a minimal product for the presentation, as this was successful we stuck to this at met up at least 3 times a week until the end of the project. Because we decided to keep this up this meant we were working long hours to keep the progress up.

For the long hours and hard work, I am proud and believe that our game, "Four Kingdoms of Kralpridan", was an overall success. This is due to completing everything we set out to do and adding the extra features to the game to make it a more playable game. From our work ethics as a group to stick together and complete this project then keep working to add the extra features and iron out the smaller parts. Due to this I feel we have done well in all aspects of the project and produced a quality end product.

10.2 Alexander Coleman

10.2.1 Overview

At the beginning of the project I had never spoken to any of my team mates so had no idea how they worked or how strong their programming was, which I think was a good because I had no preconceptions about the others. I think it also meant we wanted to work together in the labs as we didn't know each other's work ethics yet. Due to this we spent the majority of the early weeks working in the labs together and actually continued this throughout the project, which I think was very beneficial to our progress as any bugs could be worked on by the whole team straight away. By maintaining this work ethic we managed to avoid late night sessions and were able to stay ahead of schedule. I think working with people who have far more programming experience than me was very beneficial as I learnt new techniques and concepts such as enumerations and singleton classes. Overall I think I learnt a massive amount working on this project and have really enjoyed being able to work in a team with the others.

10.2.2 Working in a Team

At the start of the project I had never worked on a big programming project and had only really programmed when working on structured university assignments. This meant I was very unfamiliar with working on such a large project and having the freedom to take the project in a direction we wanted, both these things made me quite nervous at the beginning. However during our first meeting it became clear I wasn't the only one having these thoughts which made me feel much more comfortable.

During the first meeting we laid out some very basic plans for the project and assigned each member of the team a section of the game to write. Everyone had the opportunity to put forward their suggestions for both their role in the project and the overall game concept which I think helped us get off to a great start as everyone was happy with the planned game and their roles in its development.

We did the majority of our programming together in the computer science building, meeting up most days for a couple of hours. This way of working was, I think, the best way to get the best outcome because it meant if there were any bugs or someone got stuck the others were always there to lend a hand. It also meant there was a lot less pressure to go home and spend hours working each evening as we were clocking up the hours during the day. I think this very much suited some of the team members work ethic, mine included. Our goal was to have an actually viable game that ticked all the projects criteria by the user testing session in week seven. We reached this goal, which set us up excellently for the remainder of the project. We spent the next few weeks implementing extra features that we wanted to have in the final version like quests and a final boss while also working on the reliability of the game.

At the end of the project I think the game we have produced and the way we have worked together is something we can all be truly proud of. I am looking forward to hopefully working with Jordan, Priyan and Krishna again on future projects.

10.3 Krishna Patel

10.3.1 Overview

I thoroughly enjoyed working on this project. I have enjoyed playing games since I was a child and to be able to take part in creating my own is an amazing experience. I was able to work on high level features of the game such as creating environment objects, designing the world and working on player combat and game progression. My role gave me a better understanding of the user experience of software and the considerations that need to be made when creating a software for a specific demographic.

10.3.2 Working in a team

Working in a team provided me with some new challenges and experiences which I had never encountered before. I feel despite being new a team chosen for us which supposedly had similar skills we all brought something different to the project that we could contribute. I found that even though we did not know each other prior to the project we worked well as a team and got along outside the project.

As a team we regularly met up and worked on the game together as a result most of the game was written whilst we were all sitting together. We maintained good software engineering practices by continuously Java documenting making it easier for myself to understand classes and methods when I needed to use it. Any problems I had, I found it easy to bring up with the team and they would help me resolve them, especially in a face to face environment rather than everyone working from home. I learned many new techniques for game making as some of the members had experience creating games and I was able to practice these techniques whilst creating FKK. I was also able to contribute my software engineering skills in reports and planning where some of the other team members were not so knowledgeable about.

Based on my experience, this is the most efficient project I have worked on, we never left anything till last minute and worked at sociable hours rather than spending late nights coding. I believe this is because of how well the team worked together and understood the importance of the project. I would see myself continuing developing the game and working with the team in the future to possibly deploy it to the public.

10.4 Jordan Bell

This entire module has been an incredible experience. I loved working with the rest of the team, and have very much enjoyed their enthusiasm throughout the project. From our first team meeting, even though I had never met any of them before, we got along well and seemed to all be on the same wavelength. Talking about game ideas, roles each of us wanted to take, etc. felt very relaxed and was a great start.

Before the project, I had a bit of experience making small games, and was excited to use different design approaches I had seen in the past. After deciding on the game idea, using Game States and Game Objects felt like a suitable approach. I pitched it to the team and everyone seemed on board, so that weekend I made a very simple framework for the engine.

I began implementing the image-and-clip method for rendering. This meant interfacing with OpenGL (via the LWJGL java library) and learning its various complexities. That weekend we had a working engine, and I wrote the algorithm for grass generation to show off what it could do. I showed this to the team and walked them through the set of classes I had written, until everyone had a decent understanding of how to create in-game objects and render them on screen.

From past team experiences, I wanted to be wary of my common pitfalls. This time around, I wanted to be extra aware of other team members and give them room to operate freely, without me interfering unnecessarily. With that in mind, I volunteered to lead the GUI development. I wanted to work on something which would enable other members to work more easily, and creating a framework for them to use meant I could work side by side with the team throughout the development. My hopes were that I would finish work on the GUI by the time they needed buttons, menus, labels etc.

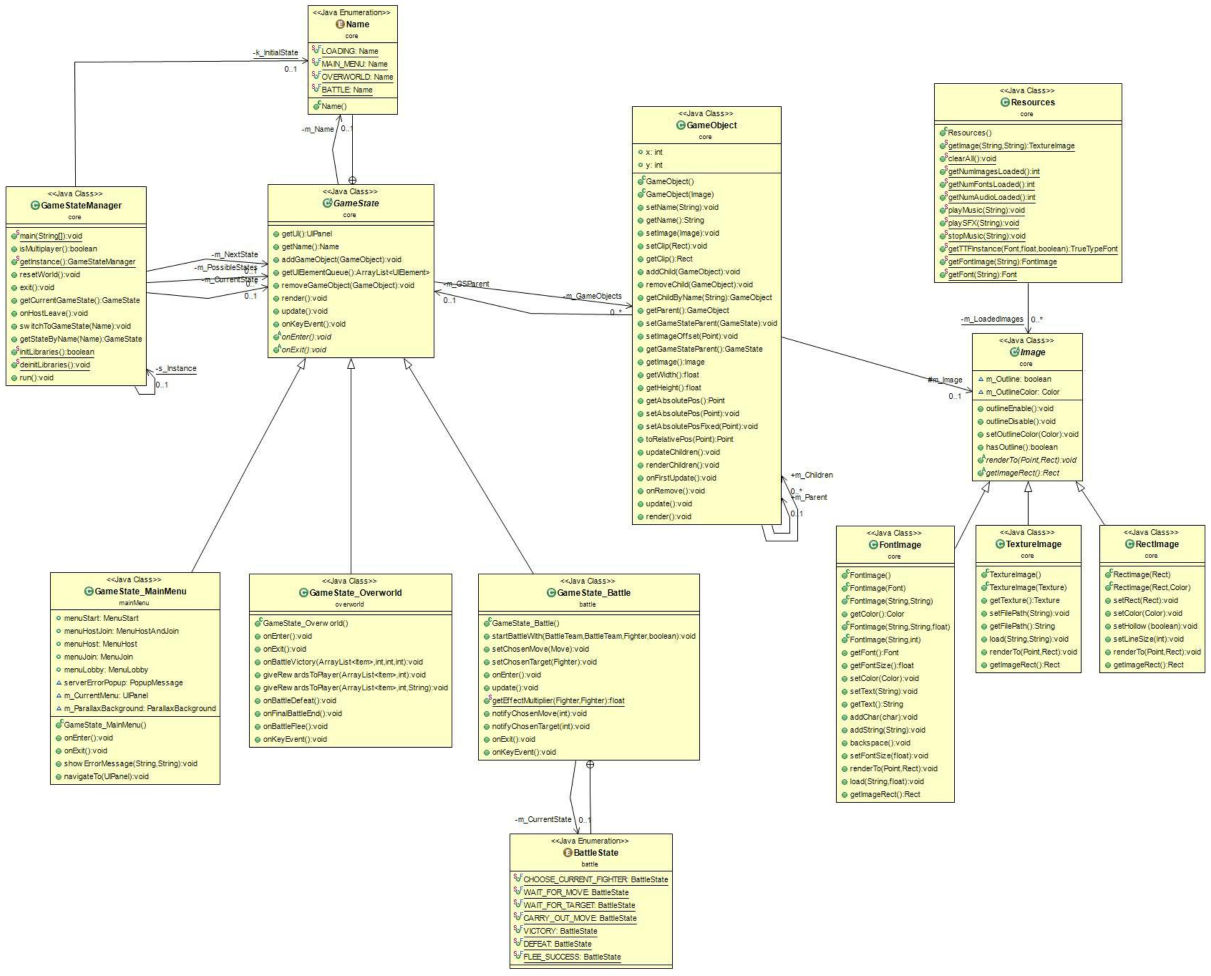
Unfortunately, prior to the prototype presentation, it seemed that I was the only one spending 20+ hours per week on the project. I finished the GUI within weeks, and was disappointed to see that practically none of the main game had been developed. During one of the team meetings, a few weeks before the prototype, I worked with Krishna on the explorable world. About a week prior to the prototype presentation, I sat down with Priyan and began the basic structure for the battle system.

After this, everyone seemed way more excited. We started talking excitedly about items, different monsters, quests, etc. which we all wanted to see implemented. After the prototype, we worked our hardest. We practically finished the game by the time that user testing game around. Everyone who played it seemed to be having fun, exploring the world, taming pets to fight beside them, etc.

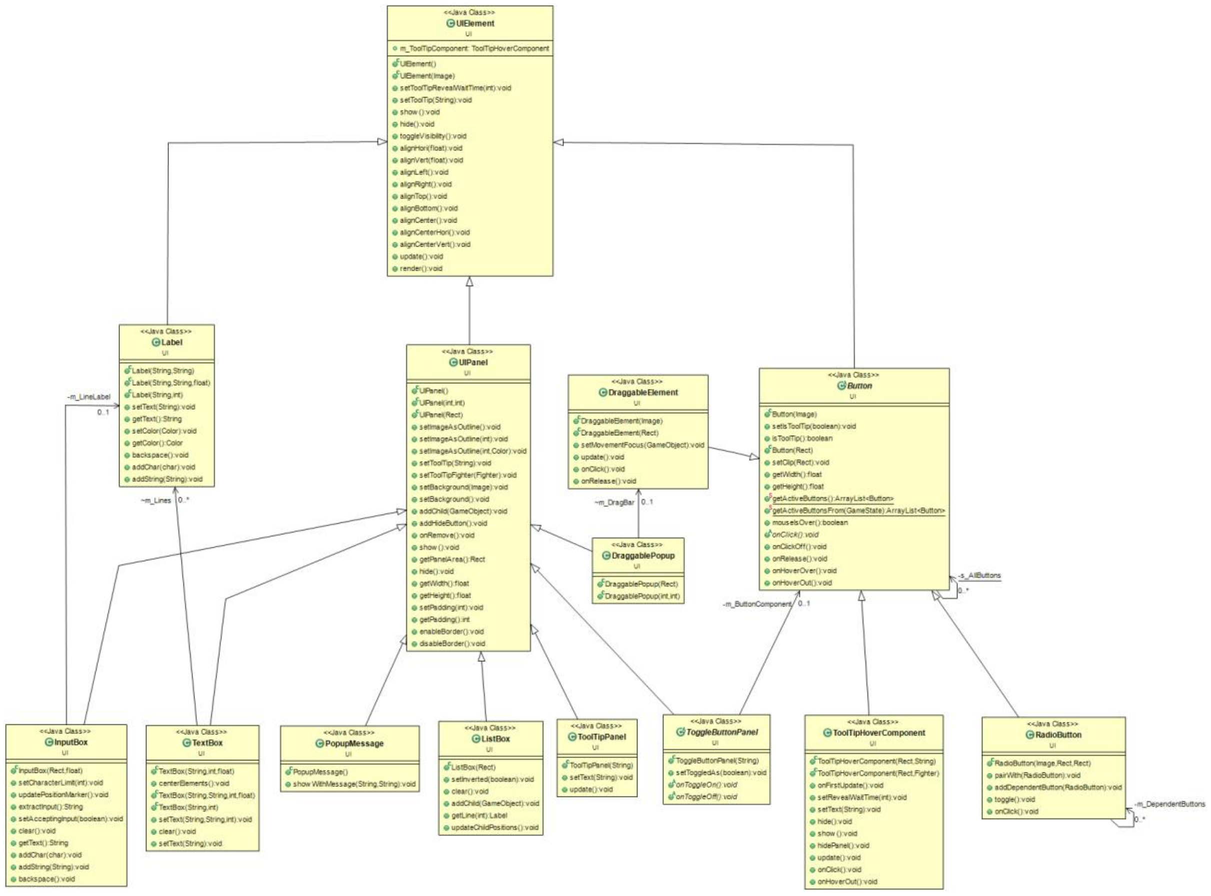
In a way, the project almost became a procrastination method. If other modules felt too hard, or if I was just bored, I'd open up eclipse and implement a mini-map, or create quests. It was awesome spending the time doing this, and the existing engine made these tasks work much more quickly.

11. Appendices

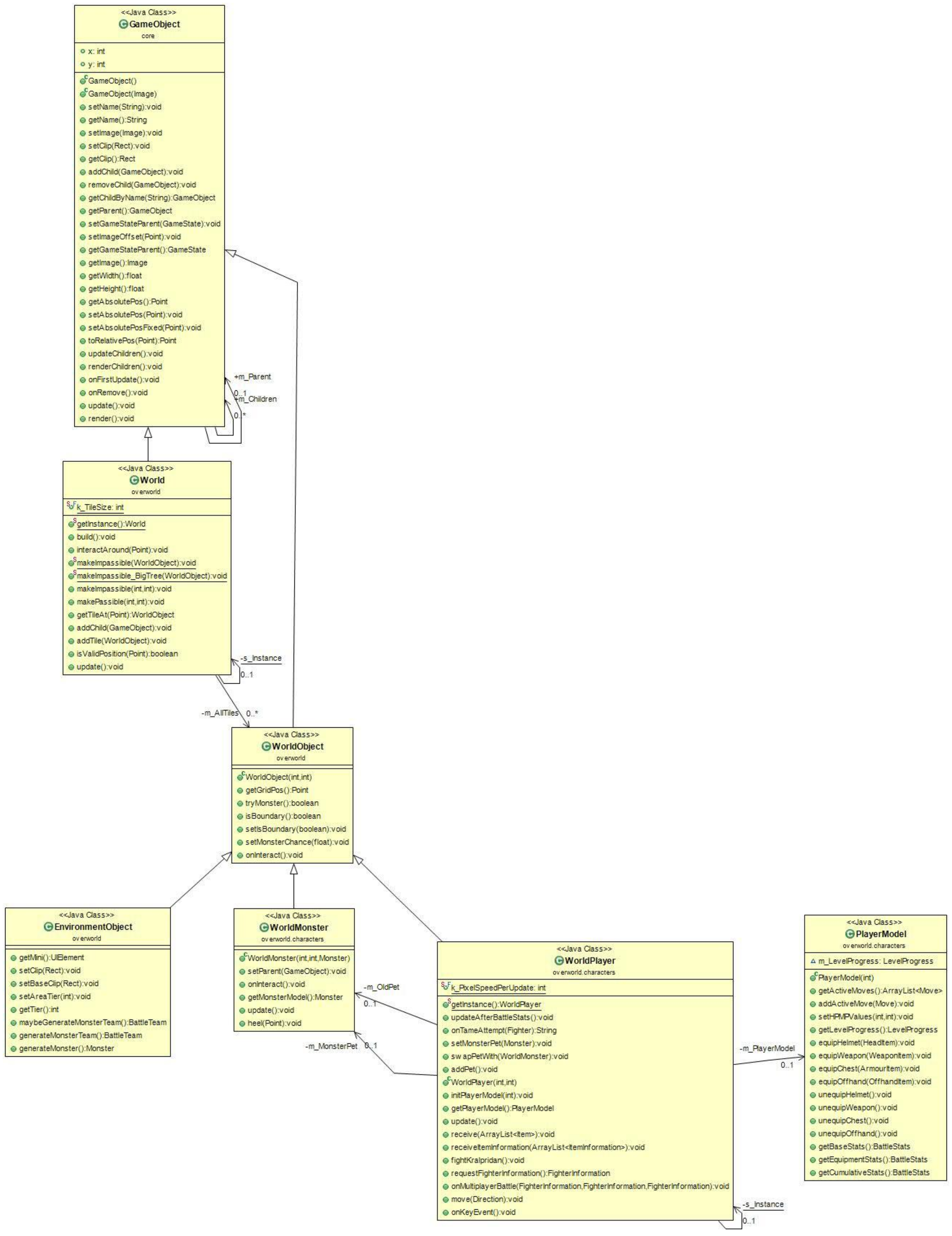
11.1 Appendix A – Game Engine UML



11.2 Appendix B – UI Framework UML

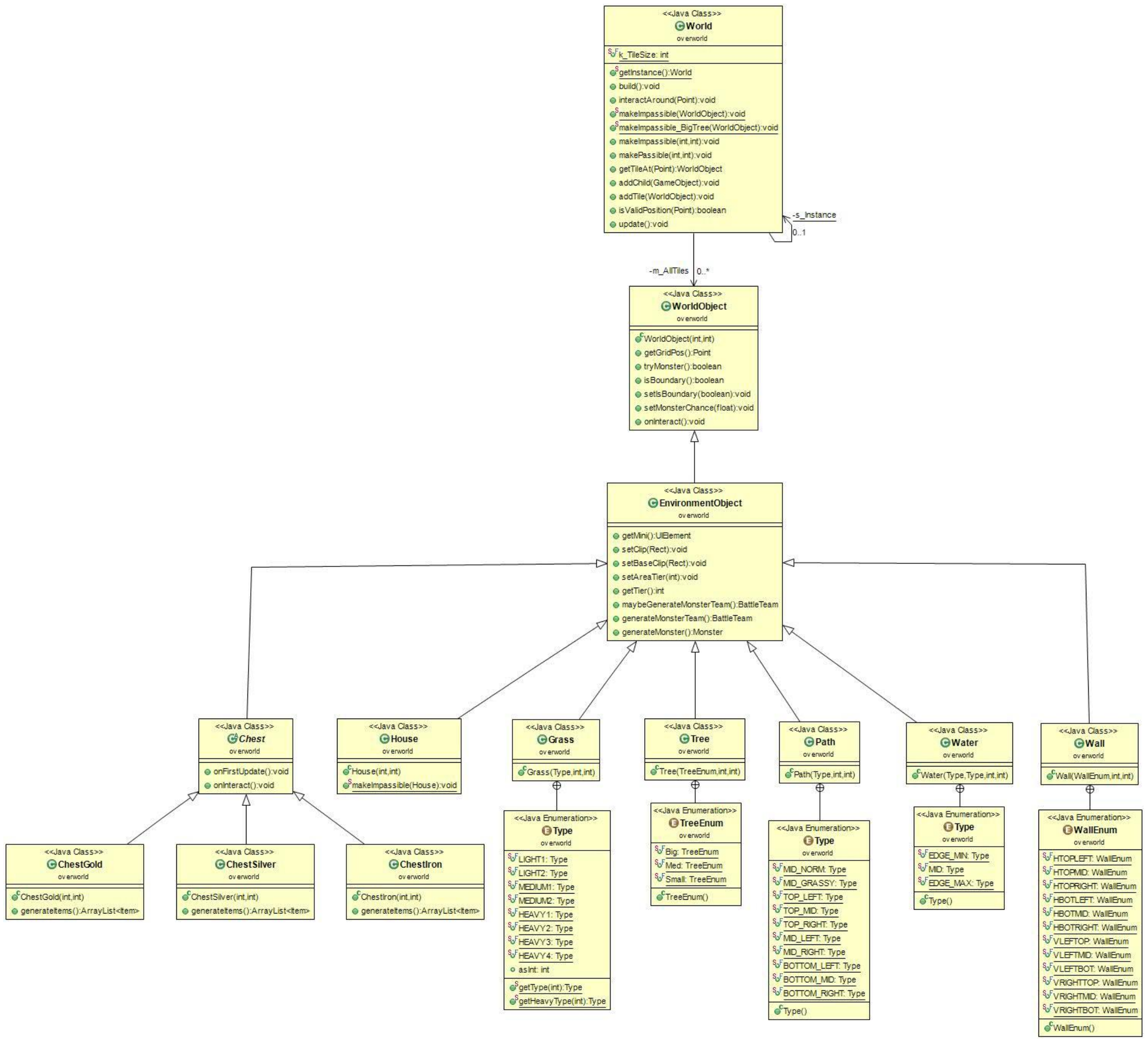


11.3 Appendix C – The Overworld UML



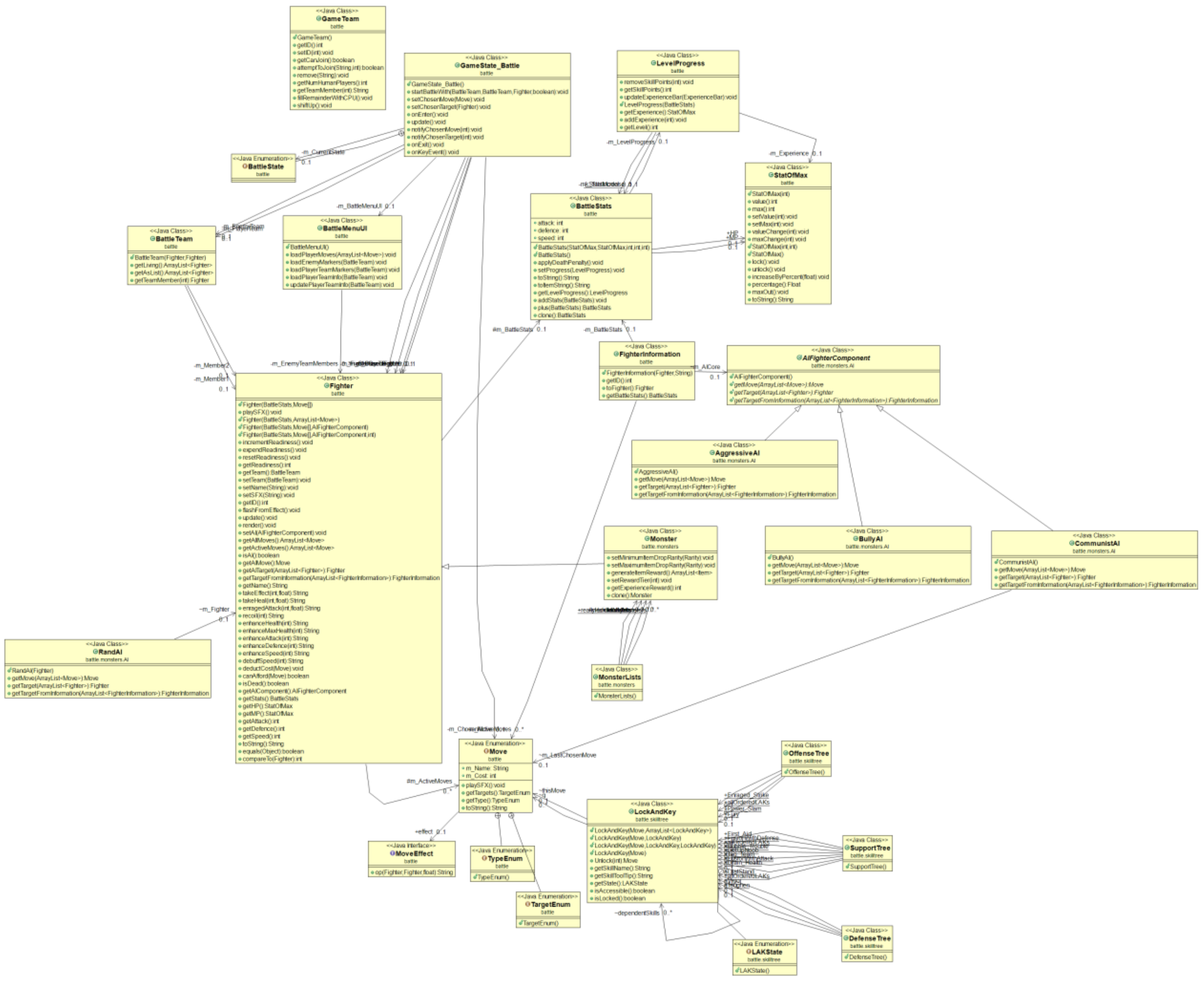
<<Java Class>>
GameObject
core
x: int
y: int
GameObject()
GameObject(Image)
setName(String):void
getName():String
setImage(Image):void
setClip(Rect):void
getClip():Rect
addChild(GameObject):void
removeChild(GameObject):void
getChildByName(String):GameObject
getParent():GameObject
setGameStateParent(GameState):void
setImageOffset(Point):void
getGameStateParent():GameState
getImage():Image
getWidth():float
getHeight():float
getAbsolutePos():Point
setAbsolutePos(Point):void
setAbsolutePosFixed(Point):void
toRelativePos(Point):Point
updateChildren():void
renderChildren():void
onFirstUpdate():void
onRemove():void
update():void
render():void
+m_Parent
0..1
+m_Children
0..*
<<Java Class>>
World
overworld
k_TileSize: int
getInstance():World
build():void
interactAround(Point):void
makeImpassible(WorldObject):void
makeImpassible_BigTree(WorldObject):void
makeImpassible(int,int):void
makePassible(int,int):void
getTileAt(Point):WorldObject
addChild(GameObject):void
addTile(WorldObject):void
isValidPosition(Point):boolean
update():void
-s_Instance
0..1
-m_AllTiles
0..*
<<Java Class>>
WorldObject
overworld
WorldObject(int,int)
getGridPos():Point
tryMonster():boolean
isBoundary():boolean
setIsBoundary(boolean):void
setMonsterChance(float):void
onInteract():void
<<Java Class>>
EnvironmentObject
overworld
getMini():UIElement
setClip(Rect):void
setBaseClip(Rect):void
setAreaTier(int):void
getTier():int
maybeGenerateMonsterTeam():BattleTeam
generateMonsterTeam():BattleTeam
generateMonster():Monster
<<Java Class>>
WorldMonster
overworld.characters
WorldMonster(int,int,Monster)
setParent(GameObject):void
onInteract():void
getMonsterModel():Monster
update():void
heel(Point):void
-m_OldPet
0..1
-m_MonsterPet
0..1
<<Java Class>>
WorldPlayer
overworld.characters
k_PixelSpeedPerUpdate: int
getInstance():WorldPlayer
updateAfterBattleStats():void
onTameAttempt(Fighter):String
setMonsterPet(Monster):void
swapPetWith(WorldMonster):void
addPet():void
WorldPlayer(int,int)
initPlayerModel(int):void
getPlayerModel():PlayerModel
update():void
receive(ArrayList<Item>):void
receiveItemInformation(ArrayList<ItemInformation>):void
fightKralpridan():void
requestFighterInformation():FighterInformation
onMultiplayerBattle(FighterInformation,FighterInformation,FighterInformation):void
move(Direction):void
onKeyEvent():void
-s_Instance
0..1
-m_PlayerModel
0..1
<<Java Class>>
PlayerModel
overworld.characters
m_LevelProgress: LevelProgress
PlayerModel(int)
getActiveMoves():ArrayList<Move>
addActiveMove(Move):void
setHPMPValues(int,int):void
getLevelProgress():LevelProgress
equipHelmet(HeadItem):void
equipWeapon(WeaponItem):void
equipChest(ArmourItem):void
equipOffhand(OffhandItem):void
unequipHelmet():void
unequipWeapon():void
unequipChest():void
unequipOffhand():void
getBaseStats():BattleStats
getEquipmentStats():BattleStats
getCumulativeStats():BattleStats

11.4 Appendix D – The Environment Object UML



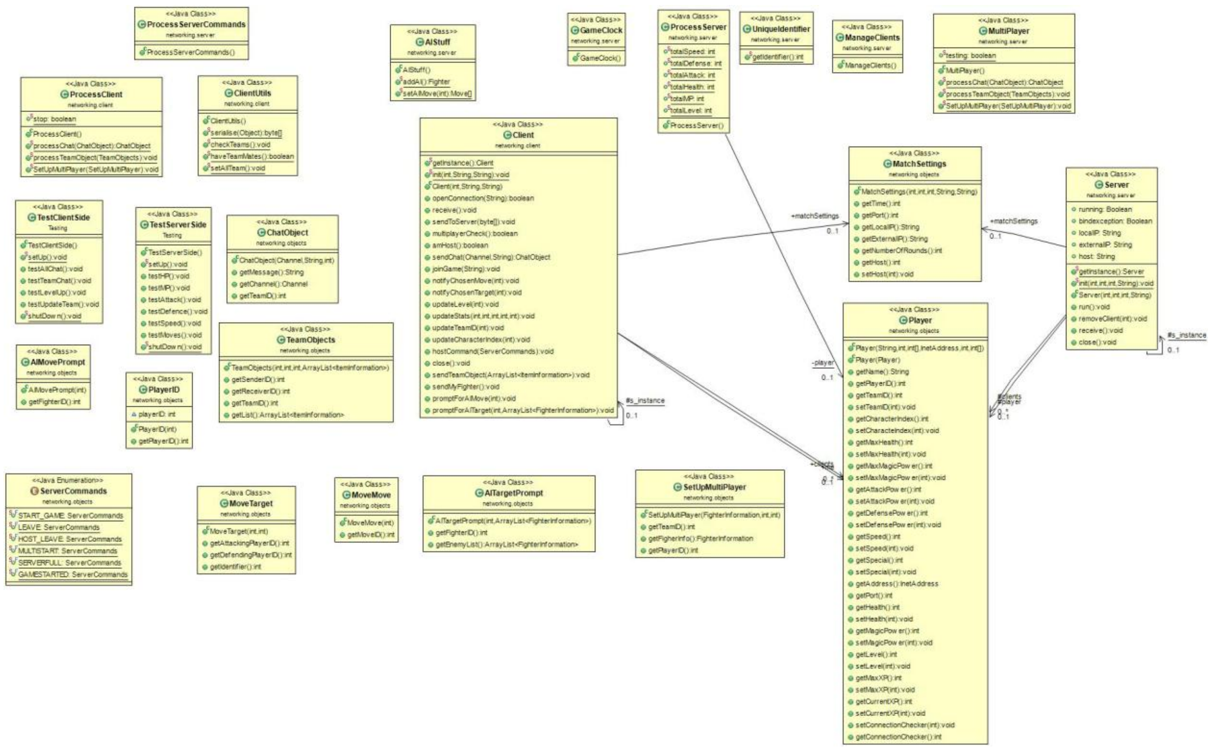
<<Java Class>> World overworld
k_TileSize: int
getInstance():World
build():void
interactAround(Point):void
makeImpassible(WorldObject):void
makeImpassible_BigTree(WorldObject):void
makeImpassible(int,int):void
makePassible(int,int):void
getTileAt(Point):WorldObject
addChild(GameObject):void
addTile(WorldObject):void
isValidPosition(Point):boolean
update():void
-s_Instance 0..1
-m_AllTiles 0..*
<<Java Class>> WorldObject overworld
WorldObject(int,int)
getGridPos():Point
tryMonster():boolean
isBoundary():boolean
setIsBoundary(boolean):void
setMonsterChance(float):void
onInteract():void
<<Java Class>> EnvironmentObject overworld
getMini():UIElement
setClip(Rect):void
setBaseClip(Rect):void
setAreaTier(int):void
getTier():int
maybeGenerateMonsterTeam():BattleTeam
generateMonsterTeam():BattleTeam
generateMonster():Monster
<<Java Class>> Chest overworld
onFirstUpdate():void
onInteract():void
<<Java Class>> House overworld
House(int,int)
makeImpassible(House):void
<<Java Class>> Grass overworld
Grass(Type,int,int)
<<Java Class>> Tree overworld
Tree(TreeEnum,int,int)
<<Java Class>> Path overworld
Path(Type,int,int)
<<Java Class>> Water overworld
Water(Type,Type,int,int)
<<Java Class>> Wall overworld
Wall(WallEnum,int,int)
<<Java Class>> ChestGold overworld
ChestGold(int,int)
generateItems():ArrayList<Item>
<<Java Class>> ChestSilver overworld
ChestSilver(int,int)
generateItems():ArrayList<Item>
<<Java Class>> ChestIron overworld
ChestIron(int,int)
generateItems():ArrayList<Item>
<<Java Enumeration>> Type overworld
LIGHT1: Type
LIGHT2: Type
MEDIUM1: Type
MEDIUM2: Type
HEAVY1: Type
HEAVY2: Type
HEAVY3: Type
HEAVY4: Type
asInt: int
getType(int):Type
getHeavyType(int):Type
<<Java Enumeration>> TreeEnum overworld
Big: TreeEnum
Med: TreeEnum
Small: TreeEnum
TreeEnum()
<<Java Enumeration>> Type overworld
MID_NORM: Type
MID_GRASSY: Type
TOP_LEFT: Type
TOP_MID: Type
TOP_RIGHT: Type
MID_LEFT: Type
MID_RIGHT: Type
BOTTOM_LEFT: Type
BOTTOM_MID: Type
BOTTOM_RIGHT: Type
Type()
<<Java Enumeration>> Type overworld
EDGE_MIN: Type
MID: Type
EDGE_MAX: Type
Type()
<<Java Enumeration>> WallEnum overworld
HTOPLEFT: WallEnum
HTOPMID: WallEnum
HTOPRIGHT: WallEnum
HBOTLEFT: WallEnum
HBOTMID: WallEnum
HBOTRIGHT: WallEnum
VLEFTTOP: WallEnum
VLEFTMID: WallEnum
VLEFTBOT: WallEnum
VRIGHTTOP: WallEnum
VRIGHTMID: WallEnum
VRIGHTBOT: WallEnum
WallEnum()

11.5 Appendix E – The Battle

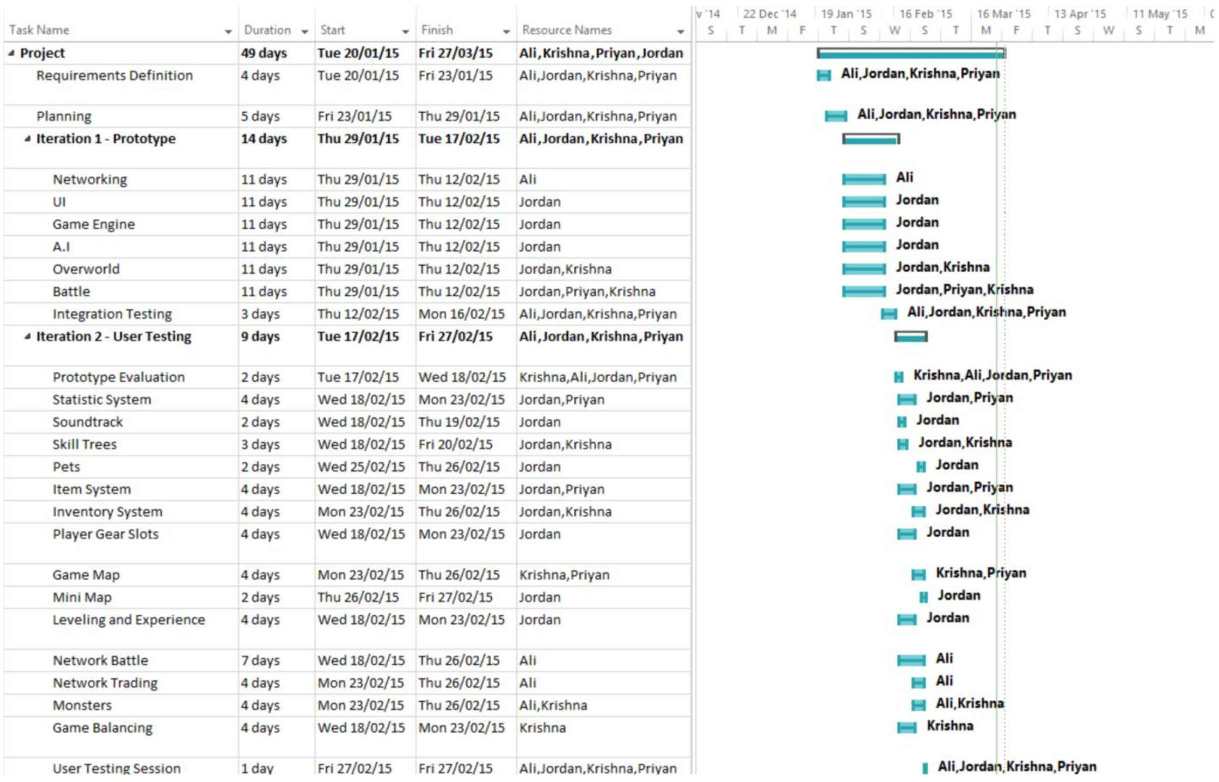


<<Java Class>> GameTeam battle
GameTeam()
getID():int
setID(int):void
getCanJoin():boolean
attemptToJoin(String,int):boolean
remove(String):void
getNumHumanPlayers():int
getTeamMember(int):String
fillRemainderWithCPU():void
shiftUp():void
<<Java Class>> GameState_Battle battle
GameState_Battle()
startBattleWith(BattleTeam,BattleTeam,Fighter,boolean):void
setChosenMove(Move):void
setChosenTarget(Fighter):void
onEnter():void
update():void
notifyChosenMove(int):void
notifyChosenTarget(int):void
onExit():void
onKeyEvent():void
<<Java Class>> LevelProgress battle
removeSkillPoints(int):void
getSkillPoints():int
updateExperienceBar(ExperienceBar):void
LevelProgress(BattleStats)
getExperience():StatOfMax
addExperience(int):void
getLevel():int
<<Java Enumeration>> BattleState battle
-m_CurrentState 0..1
-m_BattleMenuUI 0..1
-m_LevelProgress 0..1
-m_Experience 0..1
<<Java Class>> StatOfMax battle
StatOfMax(int)
value():int
max():int
setValue(int):void
setMax(int):void
valueChange(int):void
maxChange(int):void
StatOfMax(int,int)
StatOfMax()
lock():void
unlock():void
increaseByPercent(float):void
percentage():Float
maxOut():void
toString():String
+HP
+MP
0..1
<<Java Class>> BattleStats battle
attack: int
defence: int
speed: int
BattleStats(StatOfMax,StatOfMax,int,int,int)
BattleStats()
applyDeathPenalty():void
setProgress(LevelProgress):void
toString():String
toItemString():String
getLevelProgress():LevelProgress
addStats(BattleStats):void
plus(BattleStats):BattleStats
clone():BattleStats
<<Java Class>> BattleTeam battle
BattleTeam(Fighter,Fighter)
getLiving():ArrayList<Fighter>
getAsList():ArrayList<Fighter>
getTeamMember(int):Fighter
<<Java Class>> BattleMenuUI battle
BattleMenuUI()
loadPlayerMoves(ArrayList<Move>):void
loadEnemyMarkers(BattleTeam):void
loadPlayerTeamMarkers(BattleTeam):void
loadPlayerTeamInfo(BattleTeam):void
updatePlayerTeamInfo(BattleTeam):void
#m_BattleStats 0..1
-m_BattleStats 0..1
<<Java Class>> FighterInformation battle
FighterInformation(Fighter,String)
getID():int
toFighter():Fighter
getBattleStats():BattleStats
-m_AICore 0..1
<<Java Class>> AIFighterComponent battle.monsters.AI
AIFighterComponent()
getMove(ArrayList<Move>):Move
getTarget(ArrayList<Fighter>):Fighter
getTargetFromInformation(ArrayList<FighterInformation>):FighterInformation
-m_Member2
-m_Member1 0..1
-m_EnemyTeamMembers
<<Java Class>> Fighter battle
Fighter(BattleStats,Move[])
playSFX():void
Fighter(BattleStats,ArrayList<Move>)
Fighter(BattleStats,Move[],AIFighterComponent)
Fighter(BattleStats,Move[],AIFighterComponent,int)
incrementReadiness():void
expendReadiness():void
resetReadiness():void
getReadiness():int
getTeam():BattleTeam
setTeam(BattleTeam):void
setName(String):void
setSFX(String):void
getID():int
flashFromEffect():void
update():void
render():void
setAI(AIFighterComponent):void
getAllMoves():ArrayList<Move>
getActiveMoves():ArrayList<Move>
isAI():boolean
getAIMove():Move
getAITarget(ArrayList<Fighter>):Fighter
getTargetFromInformation(ArrayList<FighterInformation>):FighterInformation
getName():String
takeEffect(int,float):String
takeHeal(int,float):String
enragedAttack(int,float):String
recoil(int):String
enhanceHealth(int):String
enhanceMaxHealth(int):String
enhanceAttack(int):String
enhanceDefence(int):String
enhanceSpeed(int):String
debuffSpeed(int):String
deductCost(Move):void
canAfford(Move):boolean
isDead():boolean
getAIComponent():AIFighterComponent
getStats():BattleStats
getHP():StatOfMax
getMP():StatOfMax
getAttack():int
getDefence():int
getSpeed():int
toString():String
equals(Object):boolean
compareTo(Fighter):int
<<Java Class>> AggressiveAI battle.monsters.AI
AggressiveAI()
getMove(ArrayList<Move>):Move
getTarget(ArrayList<Fighter>):Fighter
getTargetFromInformation(ArrayList<FighterInformation>):FighterInformation
<<Java Class>> Monster battle.monsters
setMinimumItemDropRarity(Rarity):void
setMaximumItemDropRarity(Rarity):void
generateItemReward():ArrayList<Item>
setRewardTier(int):void
getExperienceReward():int
clone():Monster
<<Java Class>> BullyAI battle.monsters.AI
BullyAI()
getMove(ArrayList<Move>):Move
getTarget(ArrayList<Fighter>):Fighter
getTargetFromInformation(ArrayList<FighterInformation>):FighterInformation
<<Java Class>> CommunistAI battle.monsters.AI
CommunistAI()
getMove(ArrayList<Move>):Move
getTarget(ArrayList<Fighter>):Fighter
getTargetFromInformation(ArrayList<FighterInformation>):FighterInformation
~m_Fighter 0..1
<<Java Class>> RandAI battle.monsters.AI
RandAI(Fighter)
getMove(ArrayList<Move>):Move
getTarget(ArrayList<Fighter>):Fighter
getTargetFromInformation(ArrayList<FighterInformation>):FighterInformation
0..*
<<Java Class>> MonsterLists battle.monsters
MonsterLists()
0..*
~m_LastChosenMove 0..1
#m_ActiveMoves 0..*
<<Java Enumeration>> Move battle
m_Name: String
m_Cost: int
playSFX():void
getTargets():TargetEnum
getType():TypeEnum
toString():String
~thisMove
<<Java Class>> OffenseTree battle.skilltree
OffenseTree()
+Enraged_Strike
+allOrderedLAKs
+Power_Slam
0..1
+effect 0..1
<<Java Interface>> MoveEffect battle
op(Fighter,Fighter,float):String
<<Java Enumeration>> TypeEnum battle
TypeEnum()
<<Java Enumeration>> TargetEnum battle
TargetEnum()
<<Java Class>> LockAndKey battle.skilltree
LockAndKey(Move,ArrayList<LockAndKey>)
LockAndKey(Move,LockAndKey)
LockAndKey(Move,LockAndKey,LockAndKey)
LockAndKey(Move)
Unlock(int):Move
getSkillName():String
getSkillToolTip():String
getState():LAKState
isAccessible():boolean
isLocked():boolean
+First_Aid
<<Java Class>> SupportTree battle.skilltree
SupportTree()
~dependentSkills 0..*
<<Java Enumeration>> LAKState battle.skilltree
LAKState()
<<Java Class>> DefenseTree battle.skilltree
DefenseTree()

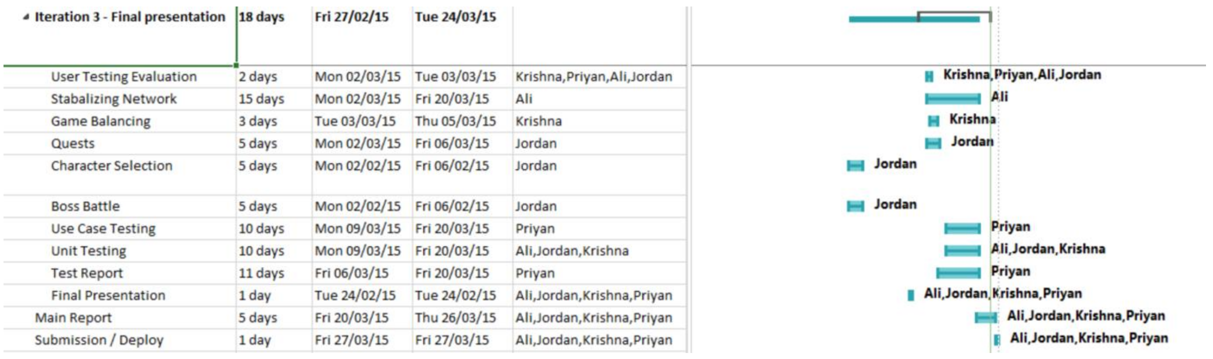
11.6 Appendix F – The Network



11.7 Appendix G -Gantt Chart



Task Name	Duration	Start	Finish	Resource Names
Project	**49 days**	**Tue 20/01/15**	**Fri 27/03/15**	**Ali,Krishna,Priyan,Jordan**
Requirements Definition	4 days	Tue 20/01/15	Fri 23/01/15	Ali,Jordan,Krishna,Priyan
Planning	5 days	Fri 23/01/15	Thu 29/01/15	Ali,Jordan,Krishna,Priyan
Iteration 1 - Prototype	**14 days**	**Thu 29/01/15**	**Tue 17/02/15**	**Ali,Jordan,Krishna,Priyan**
Networking	11 days	Thu 29/01/15	Thu 12/02/15	Ali
UI	11 days	Thu 29/01/15	Thu 12/02/15	Jordan
Game Engine	11 days	Thu 29/01/15	Thu 12/02/15	Jordan
A.I	11 days	Thu 29/01/15	Thu 12/02/15	Jordan
Overworld	11 days	Thu 29/01/15	Thu 12/02/15	Jordan,Krishna
Battle	11 days	Thu 29/01/15	Thu 12/02/15	Jordan,Priyan,Krishna
Integration Testing	3 days	Thu 12/02/15	Mon 16/02/15	Ali,Jordan,Krishna,Priyan
Iteration 2 - User Testing	**9 days**	**Tue 17/02/15**	**Fri 27/02/15**	**Ali,Jordan,Krishna,Priyan**
Prototype Evaluation	2 days	Tue 17/02/15	Wed 18/02/15	Krishna,Ali,Jordan,Priyan
Statistic System	4 days	Wed 18/02/15	Mon 23/02/15	Jordan,Priyan
Soundtrack	2 days	Wed 18/02/15	Thu 19/02/15	Jordan
Skill Trees	3 days	Wed 18/02/15	Fri 20/02/15	Jordan,Krishna
Pets	2 days	Wed 25/02/15	Thu 26/02/15	Jordan
Item System	4 days	Wed 18/02/15	Mon 23/02/15	Jordan,Priyan
Inventory System	4 days	Mon 23/02/15	Thu 26/02/15	Jordan,Krishna
Player Gear Slots	4 days	Wed 18/02/15	Mon 23/02/15	Jordan
Game Map	4 days	Mon 23/02/15	Thu 26/02/15	Krishna,Priyan
Mini Map	2 days	Thu 26/02/15	Fri 27/02/15	Jordan
Leveling and Experience	4 days	Wed 18/02/15	Mon 23/02/15	Jordan
Network Battle	7 days	Wed 18/02/15	Thu 26/02/15	Ali
Network Trading	4 days	Mon 23/02/15	Thu 26/02/15	Ali
Monsters	4 days	Mon 23/02/15	Thu 26/02/15	Ali,Krishna
Game Balancing	4 days	Wed 18/02/15	Mon 23/02/15	Krishna
User Testing Session	1 day	Fri 27/02/15	Fri 27/02/15	Ali,Jordan,Krishna,Priyan



Task Name	Duration	Start	Finish	Resource Names
Iteration 3 - Final presentation	**18 days**	**Fri 27/02/15**	**Tue 24/03/15**	
User Testing Evaluation	2 days	Mon 02/03/15	Tue 03/03/15	Krishna,Priyan,Ali,Jordan
Stabalizing Network	15 days	Mon 02/03/15	Fri 20/03/15	Ali
Game Balancing	3 days	Tue 03/03/15	Thu 05/03/15	Krishna
Quests	5 days	Mon 02/03/15	Fri 06/03/15	Jordan
Character Selection	5 days	Mon 02/02/15	Fri 06/02/15	Jordan
Boss Battle	5 days	Mon 02/02/15	Fri 06/02/15	Jordan
Use Case Testing	10 days	Mon 09/03/15	Fri 20/03/15	Priyan
Unit Testing	10 days	Mon 09/03/15	Fri 20/03/15	Ali,Jordan,Krishna
Test Report	11 days	Fri 06/03/15	Fri 20/03/15	Priyan
Final Presentation	1 day	Tue 24/02/15	Tue 24/02/15	Ali,Jordan,Krishna,Priyan
Main Report	5 days	Fri 20/03/15	Thu 26/03/15	Ali,Jordan,Krishna,Priyan
Submission / Deploy	1 day	Fri 27/03/15	Fri 27/03/15	Ali,Jordan,Krishna,Priyan

11.8 Appendix H – References and Sources

11.8.1 Network Tutorials

- Cherno Chat YouTube tutorial
    - https://www.youtube.com/playlist?list=PLlrATfBNZ98cCvNtS7x4u1bZOS5FBwOSb
    - The first version of the network was based heavily on code from these tutorials, however future iterations where almost started from scratch without the use of the videos, but there are some methods that feature in the tutorials and our final game.

11.8.2 Art Tilesets (All from opengameart.org):

- Cave: http://opengameart.org/content/db32-cave-tileset
- Weapons, characters, etc.: http://opengameart.org/content/32x32-fantasy-tileset
- Signs etc: http://opengameart.org/content/lpc-signposts-graves-line-cloths-and-scare-crow
- Tree variations: http://opengameart.org/content/tree-variations-from-jetrels-wood-tileset
- General environment: http://opengameart.org/content/2d-lost-garden-tileset-transition-to-jetrels-wood-tileset

11.8.3 Music (All from www.newgrounds.com)

- "Electrodynamix" by dj-Nate [http://www.newgrounds.com/audio/listen/368392]
- "Richwater" by Voidtek [http://www.newgrounds.com/audio/listen/605312]
- "Four Brave Champions" by DavidOrr [http://www.newgrounds.com/audio/listen/90433]
- "Final Encounter" by DavidOrr [http://www.newgrounds.com/audio/listen/205308]
- "Rage of the Champions" by MaestroRage [http://www.newgrounds.com/audio/listen/90532]

11.8.4 Sound Effects

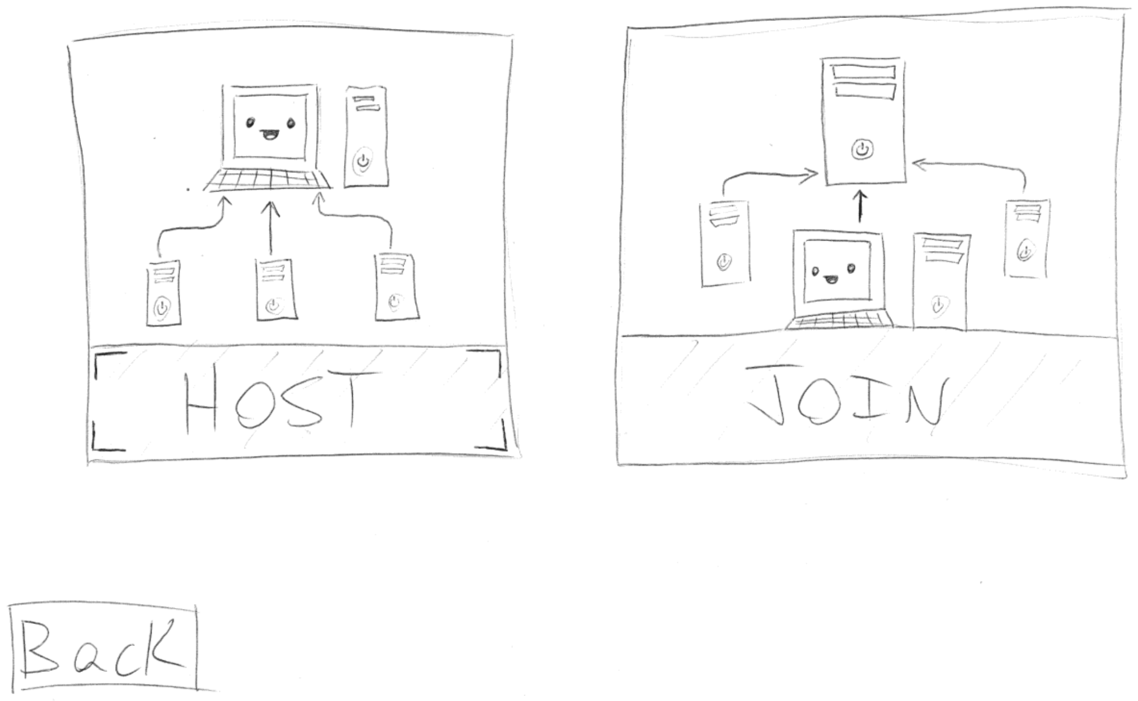
All sound effects were taken from the GDC Game Audio Bundle, courtesy of www.sonniss.com. More information at http://www.sonniss.com/sound-effects/free-download-game-audio/

11.8.5 Libraries

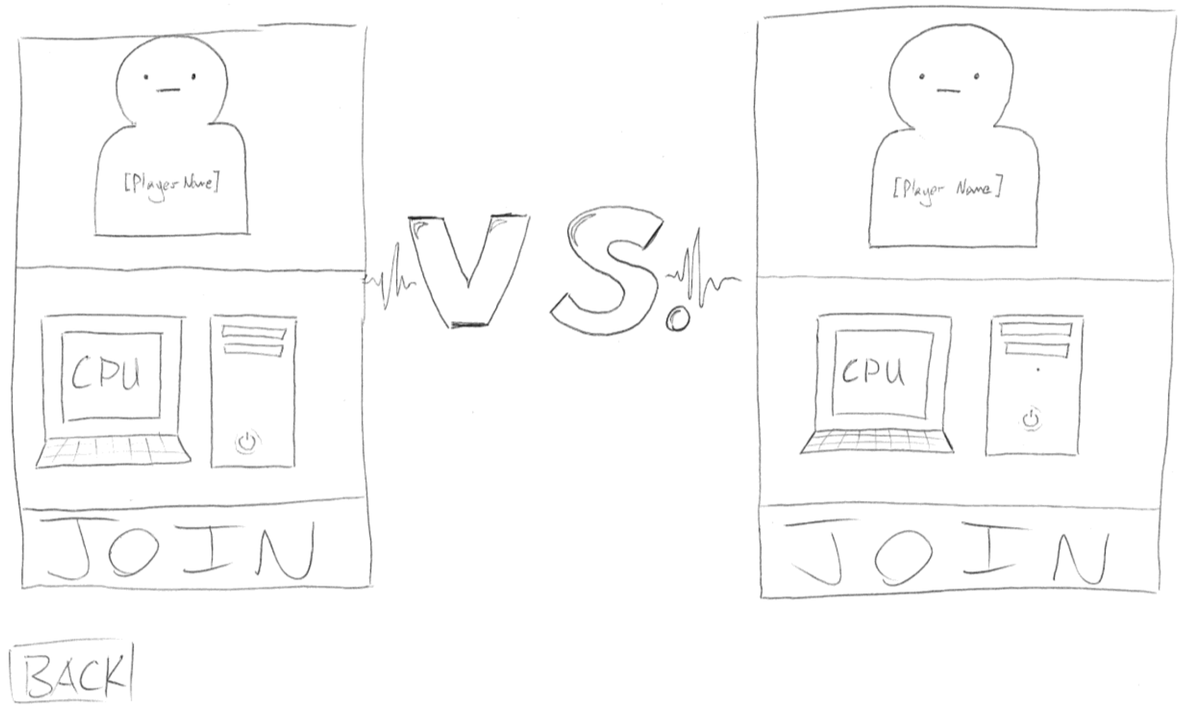
- LWJGL [www.lwjgl.org]
- Slick-Util [http://slick.ninjacave.com/slick-util/]

11.9 Appendix I - Lo-Fi Prototypes

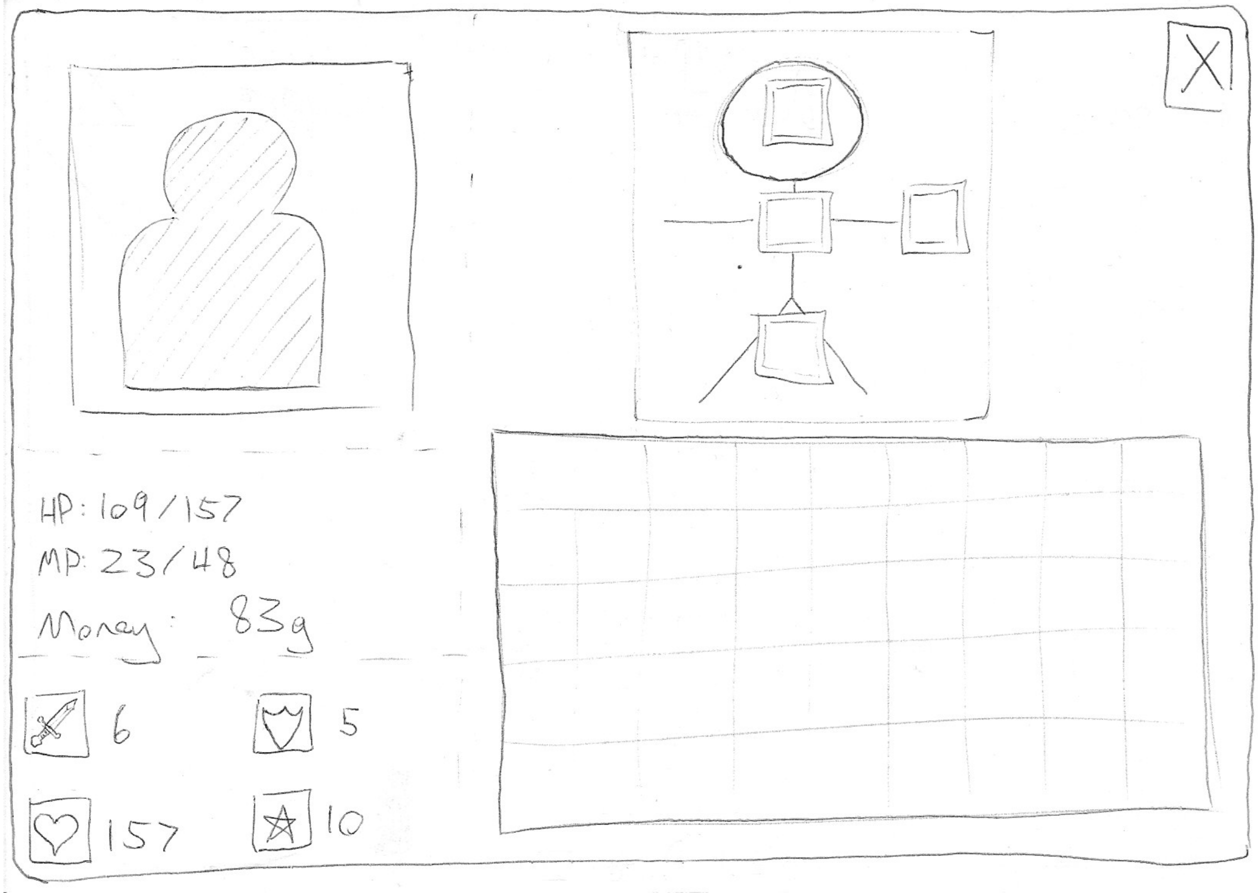
Host/Join Selection Prototype



Lobby Screen Prototype

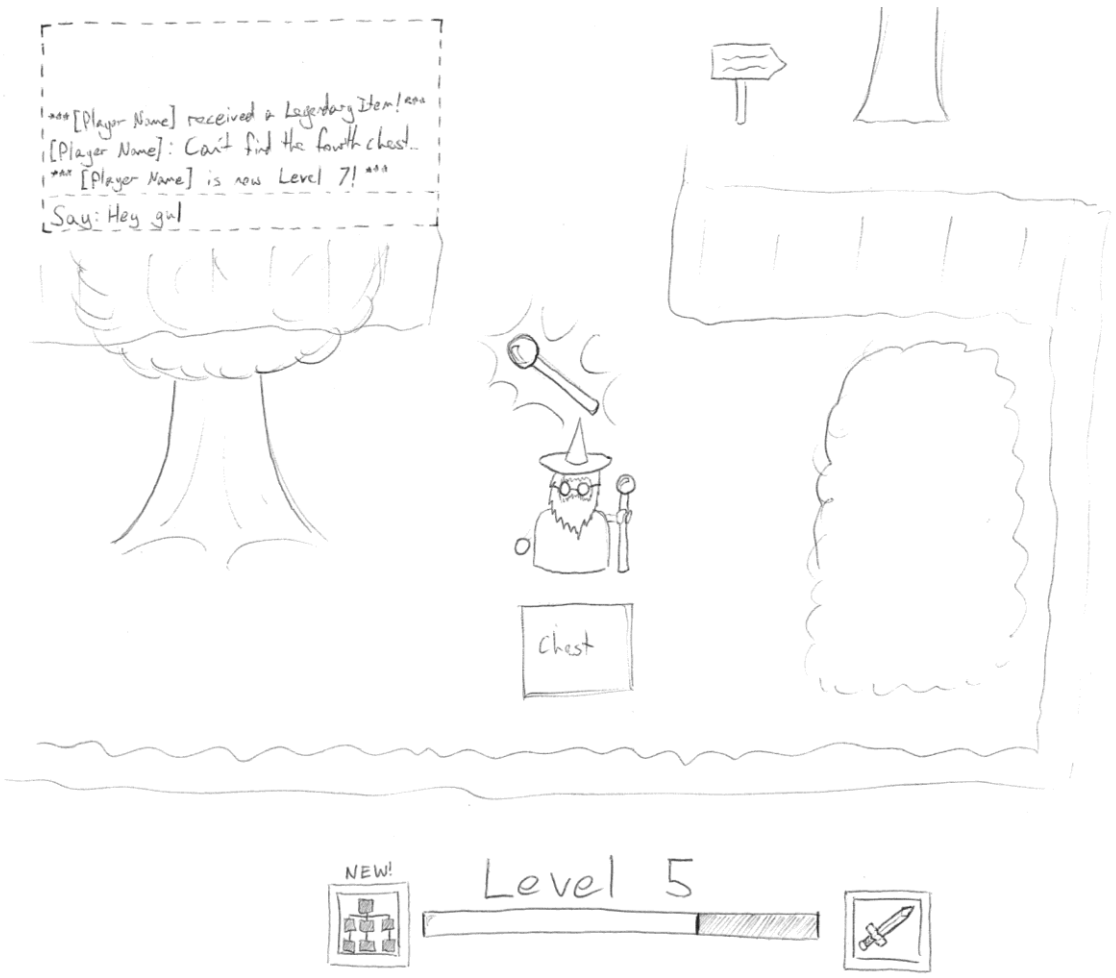


Inventory Screen Prototype

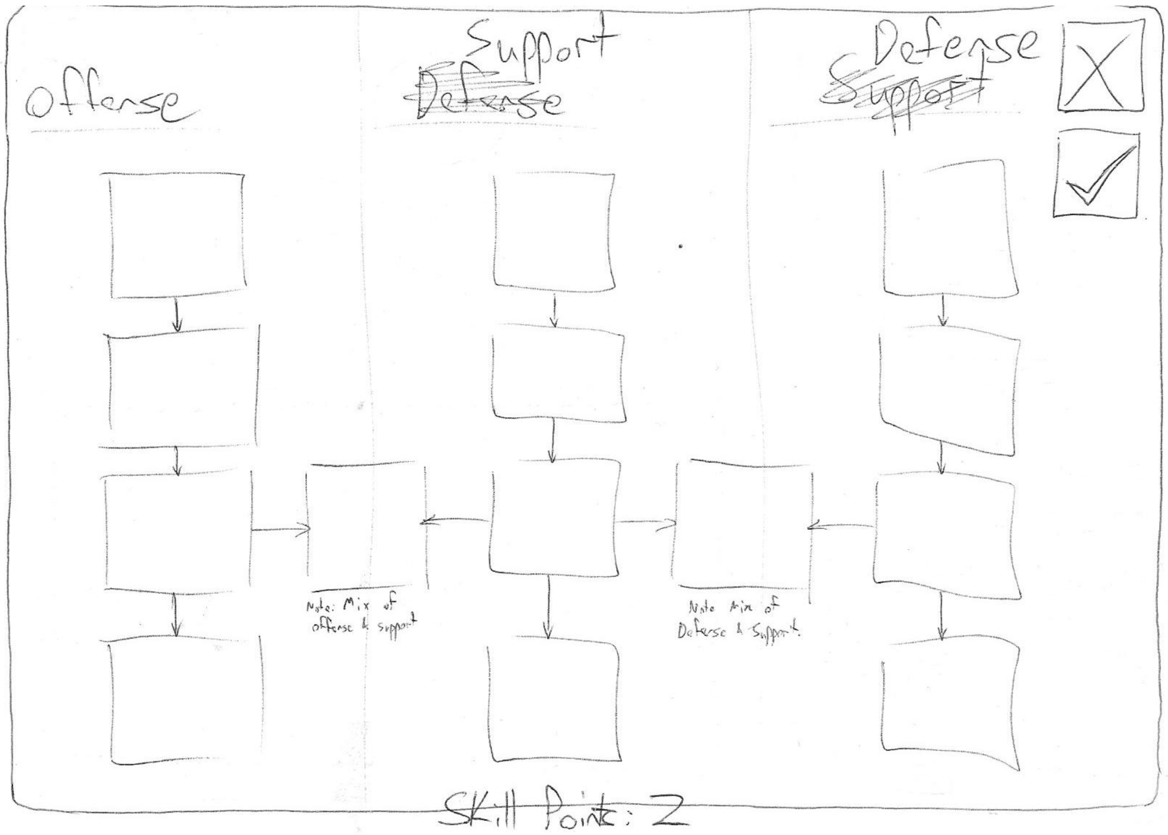


I

World Exploration UI Prototype


[Player Name] received a Legendary Item!
[Player Name]: Can't find the fourth chest...
*** [Player Name] is now Level 7! ***
Say: Hey gul

Skill Tree Menu Prototype

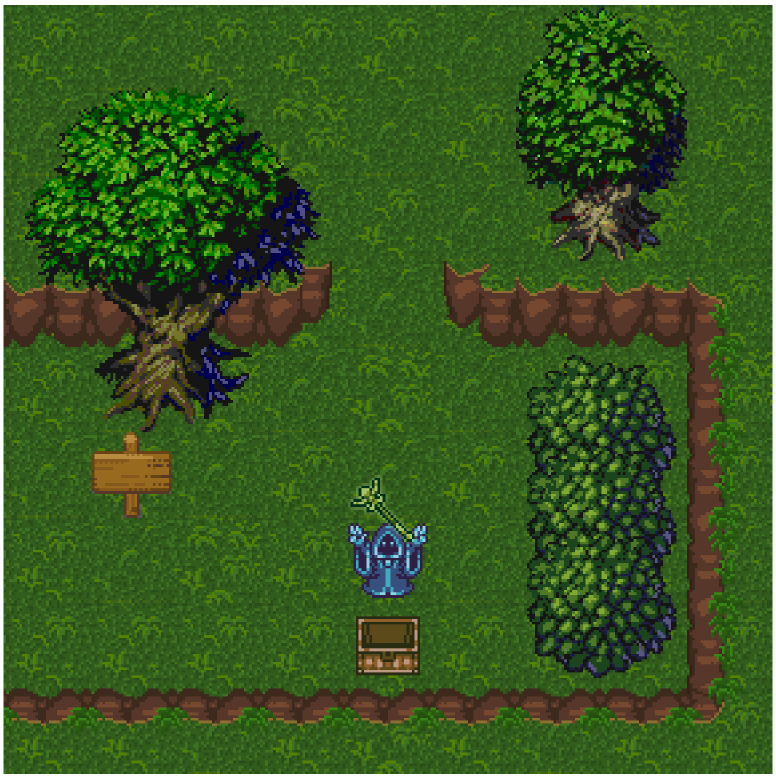

Offense
Support
~~Defense~~
Defense
~~Support~~
Note: Mix of offense & support
Note: Mix of Defense & Support
Skill Points: 2

11.10 Appendix J – Risk Strategy

Risk	Strategy	Strategy Type
Computer Breakdown	All the code that we will be writing will be saved and backed up to SVN servers. In the case of a computer problem a team member can continue to work on the CS lab computers which have the software available to continue the project.	Impact Reduction
Server Issues	Every member of the team will have a local copy of the game when they update from the SVN servers so if there is an issue with SVN we can continue working locally or transfer our project to another repository.	Impact Reduction

Team Member Illness	For each functionality of the game more than 1 team member will have understanding of the code so that if there is an issue with any particular team member another can take over.	Impact Reduction
Unable to integrate different parts of the system	The game will be written in a way that each functionality is always working with the remaining system. We will have Junit tests so ensure the correct data is being produced.	Probability Reduction
Unable to implement a piece of functionality due to time constraint or design constraint	The game will be in a working state before each functionality is added to it so we can revert back to a previous state of the game if we are unable to integrate a new piece of functionality	Impact Reduction
Networking difficulties during presentation	We have been using our own network router to run the game to prevent any issues rising if we used the university wifi network.	Risk Transfer

11.11 Appendix K - Hi-Fi World Screenshot



11.12 Appendix L - Hi-Fi Water and Paths Screenshot