DevWars

Final Report

Team B2

ID	Name
1586272	Raymond Yiu Wai Chu
1550687	Atanas Harbaliev
1533565	Samuel Tebbs
1527549	Christian Iliev
1525269	Lucy Mills
1555537	Aidan Goodwin

Contents

Introduction

This report aims to document the development of the office-simulation game DevWars. An analysis of the software and interface design employed during the development process will be presented, along with details on team collaboration, individual contribution and testing. The report will further entail an evaluation of the project's strengths and weaknesses. Attached as an appendix to this report are further useful documents, such as textual use cases, UML class diagrams and a list of test cases amongst others.

Task

We were entasked with the design and development of a multiplayer game. We wanted to create something that was different than the highly-revered but also simplistic 2D games of the past, where a player's only objective is to move around. The idea for an office-simulation game came up during the initial team meetings, it was our shared opinion that this provided vast potential for the creation of a multifaceted game with plenty of interaction with both the environment and other players. It had to feature a competitive element so as to keep the player engaged.

Similar games exist - one that bears a slight resemblance to DevWars, and in fact was a source of ideas and motivation, is EA's The Sims series, which puts the player in charge of how their life unfolds through a series of interactions with the surrounding environment.

Game Overview

DevWars is a 4 player multiplayer game, where each player is to complete a programming project before the other players are able to complete theirs. There are various actions available to the players: increasing their own project's completion rate, hindering others'. The game uses a top down 2D art style and is typically about 10 minutes long, with a single player winning, and the other players being ranked by the percentage of their completion.

The game starts when four players have connected to the server, or after a period of time, at which point AI players are added to fill any empty slots. All the players are placed in the same environment, can see the other players moving around and can interact with each other and objects in the office.

By interacting with objects in the office players can take several actions, some involve direct competition with other players, and some may be cooperative:

- **Work** *(through interacting with own computer)* - increases project completion.
    - The first minigame is Hangman, word guessing game, where the player has 10 attempts to guess a word, given its length. If they are successful, progress is added.
- **Hack** *(through interacting with other player's computers)* - hinders project completion of another player whilst aiding the current one's.
    - The other minigame is a version of the classic hit Pong, a simple "tennis like" game featuring two paddles and a ball, where the goal is to defeat your opponent by being the first one not to miss a ball, a player wins a point once the opponent misses a ball. The game is played with two human players [1].
- **Coffee Break** *(through interacting with coffee machine)* - reduces fatigue and increases productivity, however there is a chance to fail, with negative consequences.
- **Nap** - a risk-free way of reducing fatigue

External Libraries and Resources

Throughout the development of DevWars, external libraries have been utilised, in order to aid in the production of a highly functional product in a limited time:

Slick2D

We make limited use of the Slick2D game library, which provides a set of tools and utilities wrapped around LWJGL and OpenGL bindings; it includes support for images, animations which aid in the game's development. However there are a lot of functions provided by this library which we are not using, and have instead implemented ourselves.

License: Redistribution and use in source and binary forms, with or without modification, are permitted. http://slick.ninjacave.com/license/
Source: http://slick.ninjacave.com

JOGG 0.0.7

License: The MIT License (MIT)
Source: https://github.com/echocat/jogg

JORBIS 0.0.15

a pure Java Ogg Vorbis decoder.
License: LGPL
Source: http://www.jcraft.com/jorbis/

Resource sources

Menu and Game Scene Music: http://ericskiff.com/music/
License: Creative Commons Attribution License

Sound Effects: http://www.bfxr.net/
License: Apache Licence 2.0

Final Requirements

Functional Requirements

- The system shall be able to communicate over local networks.
- The system shall not need to interact with any outside services, it will be self contained.
- Four players shall be able to play at the same time in each game.
- The system shall allow for competition between players, with a single victor for each game.
- Each game shall last less than 10 minutes.
- The system shall be able to make use of sound and graphics to create an entertaining and immersive experience for the players.
- Most of the processing for the game shall be done on a single server, so as to limit the impact of users modifying the local version of the game for their own ends.
- All players shall be able to easily view each other's game progress.

Non-functional Requirements

- Adaptability: The product is expected to run under Windows 7, 10 and Unix environments.
- Reliability: The use of the system will be for recreation and so it will not be imperative to have 100% uptime of service.
- Usability: All game UI elements will be unobtrusive and not take away the focus off the game itself.
- Usability: The players are provided with options to turn up/down the sound and music volumes.
- Usability: The players must have an option to switch between display modes (full screen and windowed mode) both in the Options menu and during the game.
- Usability: Menu buttons are to give visual feedback to the user when selected.
- Accessibility: The maximum response time between click and reaction must be less than 300 ms.
- Accessibility: The maximum response time between pressing one of the movement buttons and reaction thereto must be less than 300 ms.

Software Design

The following design patterns were used during the development of DevWars, most of them occurring due to the very nature of the game e.g the Game Loop and the Update patterns, along with the Observer pattern - also naturally present, as our game had to support multiple players via networking, thus using a Server and Client. Those patterns formed the backbone of DevWars, others were used locally in different components, as noted below:

Singleton pattern - Ensure a class only has one instance, and provide a global point of access to it.

- AI: LogicHard/LogicEasy/FireRules are classes created only once for every AI player in the game. They are accessible via a method called getLogic(), which is in the AI object.

Observer - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. An instance of that pattern is the Server class.

Decoupling Pattern - Event Queue - Decouple when a message or event is sent from when it is processed.

- Events Queue:

(Sequencing Pattern) - Game Loop - A game loop runs continuously during gameplay. Each turn of the loop, it processes user input without blocking, updates the game state, and renders the game. It tracks the passage of time to control the rate of gameplay[2].

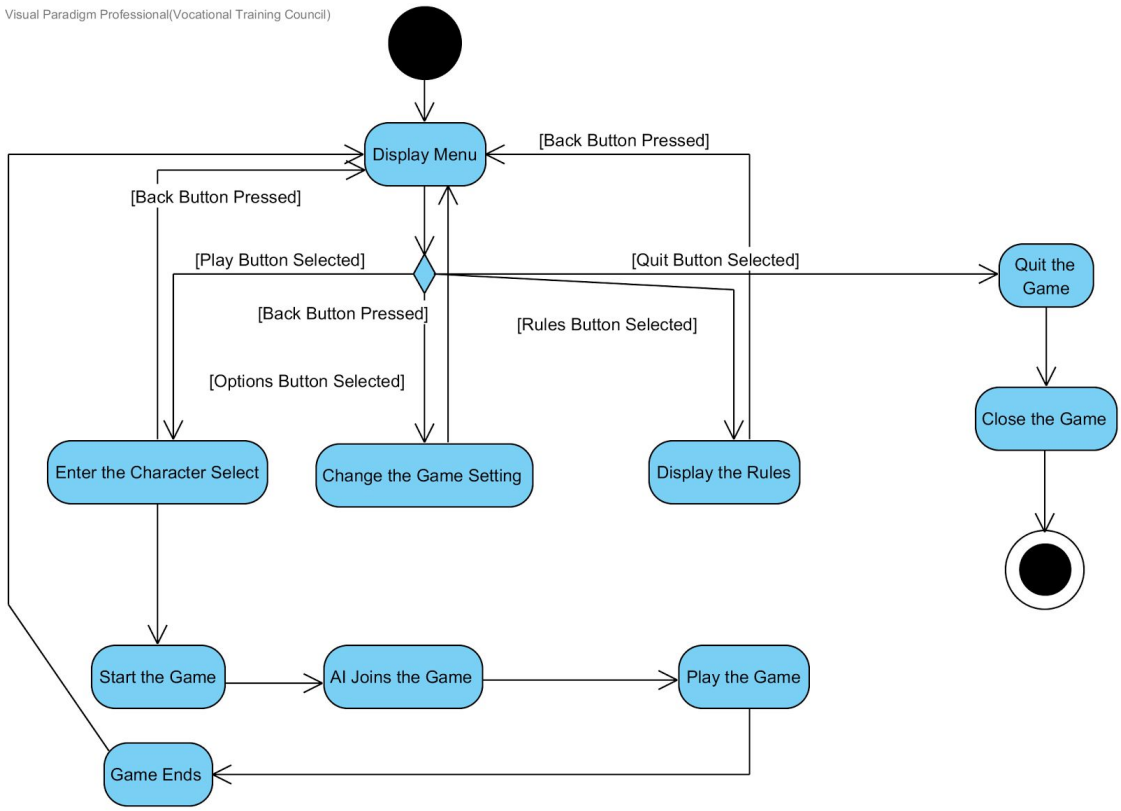
- Game logic loop: Loops at a fixed rate on the server, updating the world state (players, tiles, minigames et.c) each iteration.
- Client loop: Loops at a fixed rate on the client, detecting input and rendering the game world.

Sequencing Pattern - Update method - The game world maintains a collection of objects. Each object implements an update method that simulates one frame of the object's behavior. Each frame, the game updates every object in the collection. It works in conjunction with the game loop.

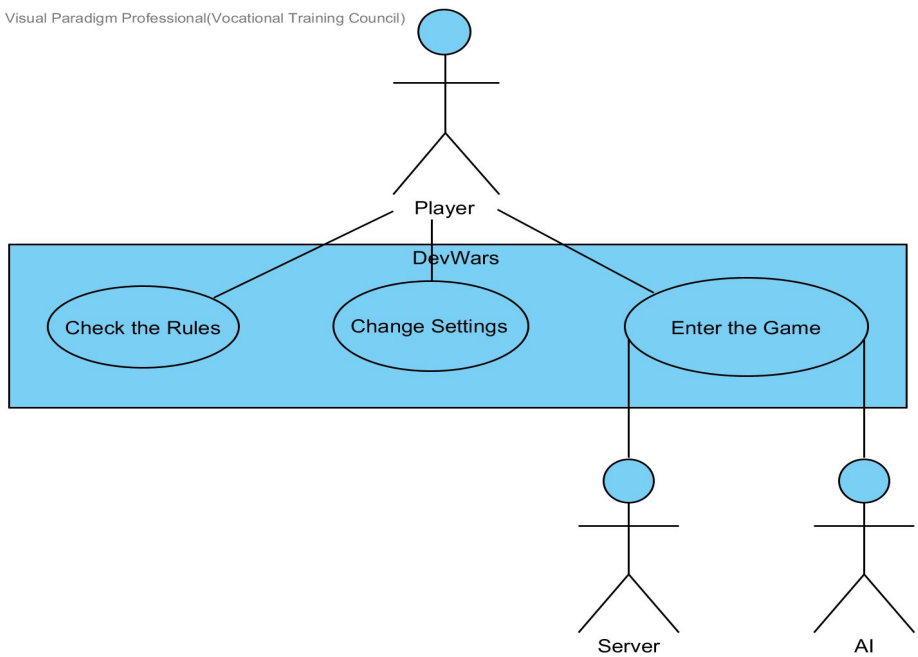
- UI: uses the update method to change how objects are rendered based on user feedback for instance.

Activity, use case and sequence diagrams (in this order) of our whole system are displayed below, along with a UML class diagram for the whole system.

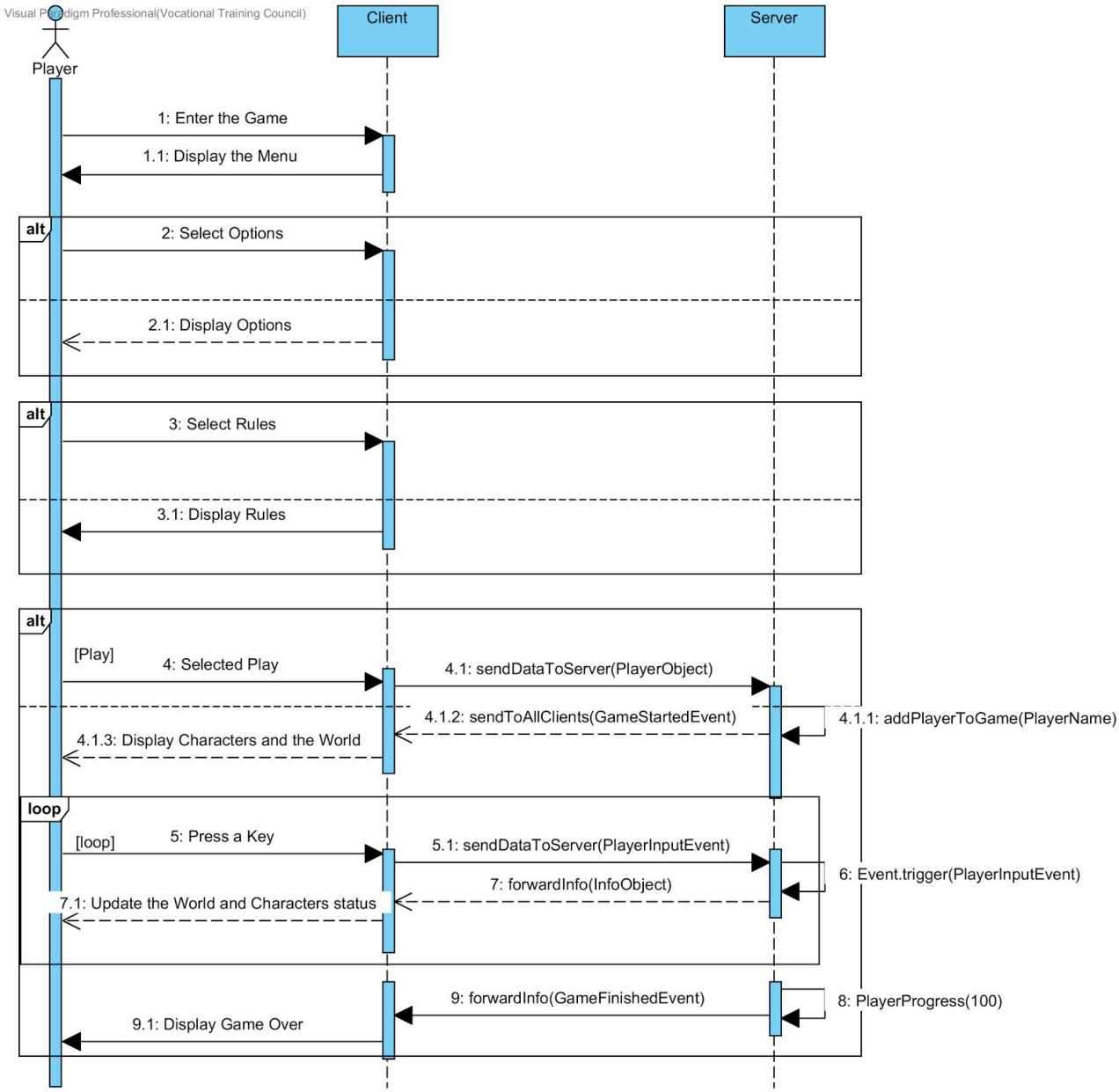
Activity Diagram:



Use Case Diagram:



Sequence Diagram:



In addition to the aforementioned:

- For UML class diagrams for the 4 main components - Game Logic, UI, AI and Networking see Appendices 8 through 12.

Main System UML Class Diagram.

Interface Design

Our primary objective is that of delivering a clean, uncluttered, and — more importantly — effective user interface.

Main Menu (fig.1)

1) Contents
    a) Upon starting the game the player is to be presented with a menu, where they can choose one of the following interactions:
        i) Play
            (1) Redirects the player to a screen where they can input a player name and server address and then wait for other players to connect. Character customisation is chosen automatically by the server.
            (2) It will automatically enter the game when enough players have been found on the network address entered.
            (3) If not, the player enters single player mode and AI players are added into the game.
        ii) Rules
            (1) Displays the rules and objectives of the game in text format along with visual cues to aid understanding.
        iii) Options
            (1) Music
                (a) The player can control the music volume in decrements and increments of 25% in the range from 0% to 100%.
            (2) Sound
                (a) The player can control the sound volume in decrements and increments of 25% in the range from 0% to 100%.
            (3) Toggle fullscreen mode/windowed mode
            (4) View key bindings - loaded separately after the player clicks “More>” at the bottom of the main options menu.
                (a) Allows the view the key bindings for movement and interactions with the game.
        iv) Quit
            (1) Quit the game.

2) Interaction - with the keyboard through the menu items itself, with the mouse inside the submenus as most of them require the player to change some settings such as the volume, along with sound to indicates that player has selected a menu item.

Figure 1 - The main menu of the game.

In Game (fig. 2)

1) Contents
    a) The main part of the screen is occupied by the game itself.
    b) The left, right and bottom sides of the screen, more specifically parts thereof are to inform the player of their and others' status and provide them with possible interactions, *whilst aiming not to obscure the game.*
    c) In keeping with the principles for unobtrusiveness, the options and pause menu are not represented by a permanent toggle in-game, but rather can be accessed by pressing the ESC key, motivated by the experience in other games.
        i) The left part of the screen is a pivotal element of the game - it shows the user's own progress towards completing the task at hand.
            (1) Since the keyboard is the primary means used to interact with the game proper itself, we have delegated the TAB key as a shortcut for this option. On pressing the TAB key, the player's own progress and fatigue levels, along

with that of the other player's is shown, along with their name and avatar, thus acting as a scoreboard on-demand.

ii) The right part of the screen provides the user with a way to view durations for the effects of certain interactions when the latter have been explicitly triggered e.g sleep due to exhaustion, coffee buzz etc.

iii) At the bottom of the screen there is a chat box where players can type messages to each other.

2) Interaction - primarily by using the keyboard - for movement, interaction, chatting, the Hangman and Pong minigames. The options menu necessitates use of the mouse.

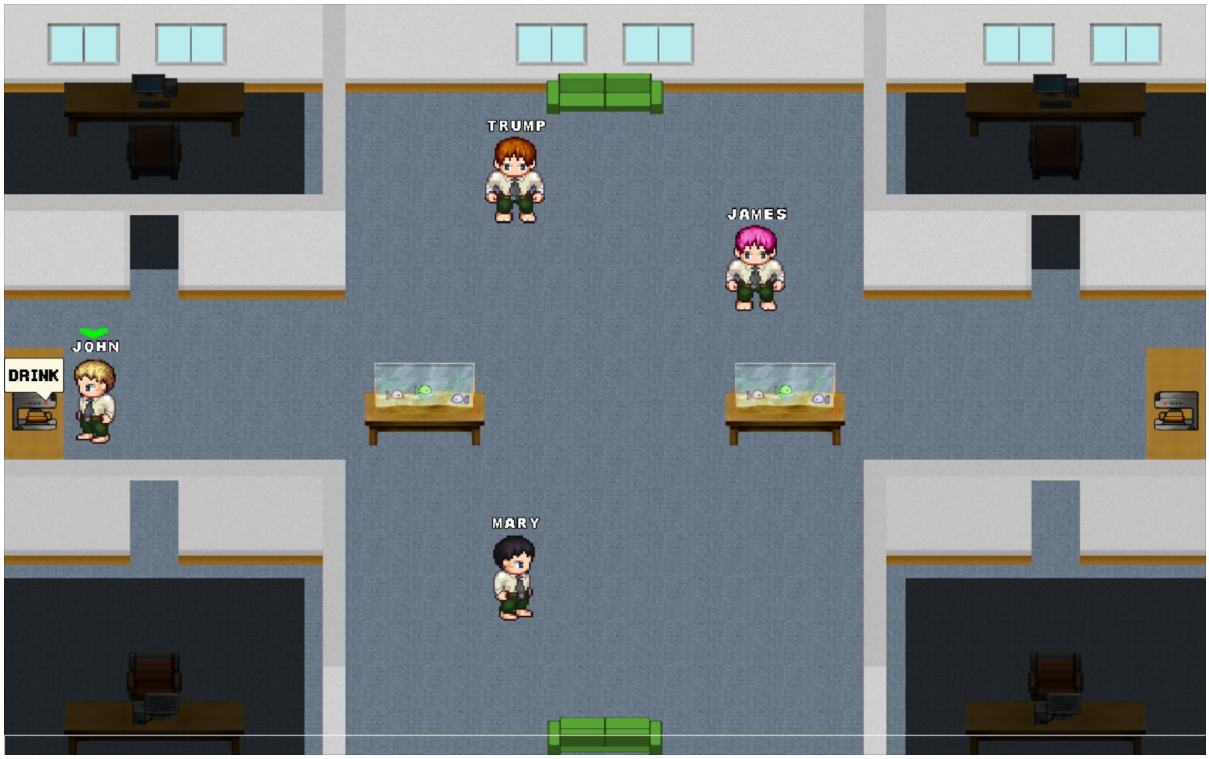


Figure 2 - The game itself with 4 players connected.

In order to achieve the aforementioned goals, usability tests were the main strategy employed to obtain feedback and make the changes players felt necessary (refer to the Testing section, UI Testing Strategy for a more in depth overview).

Software Engineering

The main software engineering principles we applied throughout developing DevWars were:

1. Separation of concerns – e.g safety, cost, performance, usability. (an example could be UI)
2. Modularity – our system is broken down into components, e.g Game Logic, Networking, User Interface, AI; This allowed us to achieve decomposability and ease of understanding.
3. Cohesion – we strived for high cohesion (each class is focused on its intention) and low coupling (changing something major in one class should not majorly affect another).
4. Change anticipation – we recognised that change is inevitable, moreover it affects all aspects of our product's development. See the Risk Analysis section (p. 16) in this report for more details.
5. Incrementality – a core element of the agile methodology – moving towards the goal of a functional game in increments.

We chose to use an Agile based methodology in the development of this project, as it offered the benefits of an incremental iterative approach, whilst also allowing for changing requirements. We had three major milestones, Week 6 - Demonstrating a working demo of the game, Week 8 - the overall game finished and Week 11 - the final release, naturally acting as successive iterations. Taking into account our desire to use a lightweight approach, viz. one that doesn't advocate too much documentation instead focusing on a functional product, in addition to our small team size and the short project lifespan, we found that the Crystal Clear Methodology[3] fitted our design and development efforts nicely. Key tenets of that include teamwork and frequent communication, as well as continuous adjustments and improvements to the process, in addition to the standard Agile process methodologies.

A core practice in Crystal Clear is the Reflection Workshop[4], requiring us to discuss the progress after milestones, looking in retrospect to find what could have been done differently and what changes we need to make for the next iteration. During the development of DevWars we met twice a week, firstly discussing what our current progress is and making sure everyone is up to speed on what has to be done. Furthermore these meetings provided a valuable opportunity, especially from the beginning of the integration process in Week 4 onwards, to discuss issues (and get help), and plan an integration strategy. It was during those reflection periods that we discovered what the main challenges that might have prevented us from not

delivering on the commitments laid out were; subsequently we were able to nominate ways of addressing those, in particular by temporarily assigning more people to collaborate on a given component and assessing whether those changes had a positive effect the following week.

Risk Analysis

We have recognised that risk management would help improve our project's success primarily by helping us anticipate and avoid likely problems and meeting the commitments we have laid out.

Our risk identification process used brainstorming, whereby we attempted to spontaneously amass ideas for the development process of our game, the product of which was a risk register table:

Fig.1 Risk Register Table

No.	Rank	Risk Event	Description	Impact	Probability	Resolution Strategy
R1	1	Architecture Complexity	- unfamiliar library in the beginning, the constraints of which are not known to us; - code base increases in size and complexity towards the end of the development process, thus pinpointing and resolving potential issues will become more complicated and obscure without the necessary understanding of the intricate interrelations between different classes.	Medium	High	- javadoc classes and methods whose function is not obvious - when integrating use the Facebook chat to seek advice from the people working on different components - discuss issues and brainstorm solutions in the weekly meetings - Pair programming
R2	2	Communication issues	- team is composed of people who have not collaborated before => this may manifest itself in communication problems in the early stages of the game's development.	High	Medium	- clearly outlined duties - weekly meetings - regular GitLab commits on corresponding branches - progress updates in the Facebook chat

R3	6	Quality tradeoffs	- time constraints in the later stages of development and a slowdown of the rate of productivity may necessitate compromises with the quality of the game. - changes made during the development may affect the system's quality attributes in an unwanted way.	Medium	Low	- Strictly adhere to the non-functional requirements laid out - constant evaluation of code by other experienced team members, refactoring when necessary.
R4	5	Requirements volatility	- requirements that we have identified in our initial specifications document are bound to change as we gain more insight into how to use the libraries.	Low	Medium	- refine the plans related to the overall design of the game. - agree on a fixed set of functional requirements that shall not be amended.
R5	4	Inadequate planning	- issues (bugs, crashes etc.) that we have not accounted for - certain features taking more than originally planned to implement	Medium	Medium	- Flexible team roles - team members shall be free to work on other areas to speed up their development after having finished with their primary tasks. - code meetings when a severe problem arises, especially if it is integration related.
R6	3	Poor time estimates (Schedule constraints)	- after Week 8 all team members would expect to have an increased workload on all other modules	High	Medium	- meeting more regularly to track progress, in addition to the Friday TA meeting, particularly in the last two weeks when bug fixing and refinement takes place, requiring the presence of all team members from their respective areas of development.

A qualitative analysis of the risks associated with the project was carried out in addition to their identification (as evident in the table above). The tool used for that was a probability/impact matrix, listing the relative probability of a risk occurring on one side and the relative impact of the risk occurring on the other:

Fig.2 Probability/Impact matrix

Probability	**high**		Risk 1	
	medium	Risk 4	Risk 5	Risk 2 Risk 6
	low		Risk 3	
		low	**medium**	**high**
		Impact		

Project Evaluation

Since the beginning of this project, one of the first design decisions that we made regarding what framework we would use has paid off- the current one the game is built upon is very flexible, thus it can be easily extended, by adding more minigames for instance. Another success is the UI, which could not have been improved without the the usability testing we conducted. It is currently intuitive and well-executed, and moreover it fits with the overall look and feel of the game we are aiming for. The element of interaction between players involves a complicated system of events, which is very extendable - new types of event can be added and integrated seamlessly, albeit it has a steep learning curve. It is our feeling that this could be improved in the future. The interaction between players provides the game with a strategic element, and this is precisely where varied gameplay stems from - players can progress both by individually working and by competing with each other, without which the experience we provide our players with would not be engaging.
The strengths of this project would not be possible without Git, which we started using since the beginning. Had we not set all branches in the very beginning, as we did, separated by components, that would have manifested itself in problems with team organisation in the long run, ultimately leading to us not meeting the deadlines. We had to revert commits and restore to previous working versions on numerous occasions, which everyone appreciated worked flawlessly.
A weakness which manifested itself in the later stages of our development is the repetitiveness of the game, partly due to the limited number of minigames, which we currently only have 2 of - Hangman and Pong. This could be rectified by making more mini games if it were not for the time constraints of the project.

From personal experience, creating a new map for the game has proven very straightforward as the creator does not need to consider the system internals, however that is as far as the object placement in the world is concerned. A further addition providing more extensive customisation options would be a level generator, where a simple interface with drag and drop functionality to place objects in the game, and toggles for number of players, map size etc. would be provided. This would be an excellent supplement to the game as it would enable players with no programming capabilities to tailor it as they see fit, abstracting away the technical knowledge of the game internals that is currently necessary for altering some of those components. Adding new visual components, such as buttons, or changing existing ones, such as strings, is also quite straightforward, owing to our in-house designed font, word generator to go along with it, and the Button interfaces.

Satisfying the functional requirements is the main goal of the project, and that we consider as accomplished, however attention also needs to be paid to the non-functional ones, most of which have to do with usability and the quality of interaction in general. The feedback we received played a key role in the improvement of the User Interface in particular. The approach we took to testing UI elements was usability testing, involving the users as much as possible. An in-depth overview of that and results obtained can be found under the UI Testing section of this report (p. 39) and Appendix 7: UI Testing. As the results of that showed, our initial designs were not met with overwhelming approval, and users found it mostly unintuitive to interact with certain game elements. This was a valuable lesson in that the developers' definitions of what can be considered as easy to use is not always the same as the users', so feedback must always be sought. It was primarily this feedback that guided us in making the changes laid out in Appendix 7. We learnt from experience that usability testing has to be applied as early as possible, immediately after a working prototype has been developed, so as to allow the development team to make changes incrementally, instead of being overwhelmed with unsatisfactory feedback when nearly all game components have been finished already thus making alterations arduous.

Whilst it is true that the separation of work and the flexibility between team members was effective in achieving the good design and usability of the system, a possible improvement that may have proven useful in the latter stages of development of DevWars is all team members having a more comprehensive knowledge of the existing code base. In the last two weeks there were issues related to threading and concurrent modification in particular, but not all team members were able to help in fixing those, due to them being preoccupied with other elements of the game, which whilst still vital, had little to do with the aforementioned issues. By ensuring that everybody has a thorough understanding of the whole system we could have used the additional manpower in fixing bugs quicker and might still have had time to work on the other important areas of the game.

This project presented us with an opportunity to apply software engineering techniques, such as Requirements Gathering and Risk Analysis, which were not merely planned and forgotten, but rather made use of during our development process. Having come up with a plan and most of the requirements at the very beginning of when we started working together has proved to be the difference between success and failure, a fact universally recognised by all of us, especially in comparison with the experience in our respective teams during the last year's team assignment in Robot Programming. A further learning outcome is applying JUnit testing to a range of classes - there is no way of uniformly testing the variety of

different components we had, in turn meaning we had to apply different testing strategies to each component, as they vastly differ in the functions they perform - AI and Networking in particular proved difficult, but with the help of other members of the team those challenges were overcome.

Altogether, DevWars can certainly be considered as a successful project, satisfying both its functional and nonfunctional requirements. The current implementation provides room for improvement and extension, which as discussed herein should not be challenging due to the inherent flexibility of the design we have adopted, with the only limitation being time.

Teamwork Evaluation

Component	Contribution per component
Game Logic	Samuel Tebbs: 95% Aidan Goodwin: 2.5% Christian Iliev: 2.5%
Networking	Lucy Mills: 95% Aidan Goodwin: 5%
Graphics/Rendering	Aidan Goodwin: 60% Christian Iliev: 15% Raymond Yiu Wai Chu: 25%
UI	Aidan Goodwin: 33.3% Christian Iliev: 33.3% Raymond Yiu Wai Chu: 33.3%
Audio	Christian Iliev: 40% Raymond Yiu Wai Chu: 60%
AI	Atanas Harbaliev: 100%

**The representation shown above is solely for code written and does not factor in bug fixing and planning/documentation.*

We were all committed to create something more comprehensive than one the classical games of the 80's and 90's. In order to achieve the high rates of effectiveness of our team, we utilised a number of tools and strategies, starting from the first week we met. Since the GitLab repository was not available at the time, we immediately set up one on Github, which we used during the first two weeks. A feature of our team-work and communication which proved integral to development is our branch model using Git. The integration branch would be reserved for stable

builds, and if we needed to work on bugs, features, or game component, a separate branch would be created. This enabled us to work independently on our own branches without interfering with anyone else's code. All branches that we use today were established during that initial period of time. On account of all the above, we had the ability to improve the working version of the game on the integration branch each week.

Facilitating communication relied upon using Facebook chat, where we would keep up to date with what team members have done, and what they plan to be doing. If anyone had issues that they needed assistance with, this group chat was the main resource to get quick help. In addition to that, we also created a Facebook group, used to set up event reminders for our weekly meetings, and post links to files and documents we are working on, so everybody could collaborate in the easiest way possible. One advantage of using the tools facebook provides for keeping in touch is that it was ubiquitous on all our team members laptops and personal devices, but what we found especially useful were the real time push notifications even when the facebook applications were not open, allowing team members to always be in touch with the latest changes and plans, without requiring any effort in remembering to open and check other particular applications that would provide a similar functionality. Thus everyone was able to get a swift reply and share their status with other team members.

From our past rather limited experience with team projects, namely the one in first year in Robot Programming, we have all realised that communication and proper planning is a key element of success. Therefore, from Week 2 onwards, we decided to meet at least once a week for two hours, typically on Tuesday, in addition to our TA Friday meetings. The fact of the matter is that we met more often, usually twice weekly, and in the last few weeks even every other day for periods of time well over two hours. Before the prototype was done in Week 6 we would book a meeting room in the library, but afterwards we were reliant on the infrastructure that the labs in the Computer Science building provided in order to implement new features and fix bugs. During our meetings we would review what has been accomplished and devise a general plan of what each team member has to do during the next iteration. We chose the Crystal Clear agile methodology, as described in Section Software Design herein (p. 7), as it embodied the principles of teamwork and continuous improvement, enabling us to examine what can be done differently for the next iteration.

Another area, that from our experiences last year was overlooked, was team bonding outside of working, which we consider had a noticeably positive effect on communication between all of us early on, seeing as we neither knew each other nor we have worked together before.

With regard to the artifacts that we needed to produce, we needed a tool that would improve our efficiency and productivity, whilst allowing everyone to work on a document at the same time. We settled on using Google Docs which fulfilled all of those requirements, and allowed us to focus on the Design Specification document and subsequently the Final Report, abstracting away the need to worry about file extension compatibility on the different systems we are using and formatting issues.

We learnt how to improve team interaction and responsiveness. During the past eleven weeks everyone was helpful and considerate towards the rest of the team. During our weekly meeting all team members were present and fully dedicated to the task at hand. Furthermore, all disagreements between people were taken care of in a peaceful and friendly manner.

Everyone equalled contributed to the project and all team members brought different skills and knowledge bases to the project. As we got on so well as a team we were able to utilise these skills fully which has allowed as to produce a game which we are all proud of and our peers also enjoy playing, which our testing has proven.

All team members were happy to help others even when the problem didn't affect the part of the code they were working on. On top of the group meeting we met in smaller groups for peer programming and to help each other fix issues and debug code.

Individual Reflections:

Team Member	Contribution
Aidan Goodwin	16.6%
Atanas Harbaliev	16.6%
Christian Iliev	16.6%
Lucy Mills	16.6%
Raymond Yiu Wai Chu	16.6%
Samuel Tebbs	16.6%

Lucy Mills' Reflection

Reflection on teamwork:

I believe we worked very well as a team. We all shared the same enthusiasm for the game idea and all wanted to make our idea work resulting in a robust game. We had a group chat which everyone was very efficient in responding too, which made it easy to ask questions, make bug fixes when we weren't together and arrange meetings.

Everyone turned up to the meeting as focused on the task. The project was equally distributed and although we all helped each other nobody took over anyone else work ensuring equal contributions.

We have met up socially as a team which helped us have a strong bond and I believe this contributed as we all got on and therefore wanted to do well in our own code to ensure it didn't affect others.

As we worked effectively as a team it meant in the last week we only had bug fixes and therefore was not in a mad panic to get it finished so didn't make silly mistakes or have to use inefficient code.

In conclusion I consider that our team was a success and we have all supported each other well and I would happily work with all of them again.

Reflection on individual contribution:

I consider my contribution to the team to be valuable. In one of our initial meetings it was agreed that I would take on the role of Networking.

It was important for me to get a working client server model in early on as it was key this was integrated as it was likely to the most integration issues and also once in place allowed more accurate testing of all the other components. I feel I was successful in this as I had a working client server model that could listen to events and send them to and from the client by our team's integration deadline, a week before the mid way review.

After this I worked on improving the server for example not allowing two clients two have the same name, adding ai players when someone dropped out etc.

An issue I came across was there were occasions where I couldn't test my additional code until other elements of the the project were complete, however this did not affect my ability to finish on time and the team worked together to fix bugs which were affecting everyone to ensure they were solved quickly. Another issue I came across was blocking methods such as the accept method, this involved some research in ways around to allow my code to work in the way I intended.

I took on networking as when we covered it in first year I didn't feel like I fully understood it. From doing this project I know have a much stronger understanding of Client-Server models and feel I could go on to write another one for larger projects.

Team Project individual contributions assessment

Your name: Lucy Mills Your team: B2

Team member (include your own name in this list)	Contribution (%) (should sum to 100%)
Aidan Goodwin	16.6%
Atanas Harbaliev	16.6%
Christian Iliev	16.6%
Lucy Mills	16.6%
Raymond Yiu Wai Chu	16.6%
Samuel Tebbs	16.6%

Raymond Yiu Wai Chu’s Reflection

Reflection on Teamwork:

The team worked and cooperated very well during the game development , we scheduled regular meetings in each week. We are usually discuss ideas and problems that we have in different stages, and beautify the whole game design bit by bit in each meeting. We frequently use Facebook group and group chat to communicate and schedule meetings.

We had a good time during the development and we met most of our aims after half of the development period, we could have added more interesting elements into the game to make it even better. But overall the whole team has done a good job on this game with a reasonable pace. We helped each other during the development even though it wasn’t the part that we are assigned when we first met, and it came along really well. We are happy with the outcome at the current stage of the game, and it can has more functionalities and fun as there is potential on the game.

Reflection on Individual contribution:

In the very first stage of development, I created a craft for the game with only has start scene and single play scene as a sandbox to test the possible elements we are going to add on in the later stage of game development. I am mainly responsible for the Graphics and Sound system, which included buttons,background images, fonts, background music, sound effects. At the beginning of the development, I have drawn the menu buttons with hover on and off effects by using Photoshop.

Later on, I found out that it would be more convenient if I could have a word generator that generate words with the font we want.Therefore, I have written a word generator that is a main feature and widely used in the game, which is generated 8-bit style words for player names, and system notifications,pop ups, settings pages and rules. So we can have an united style for the whole game rather than draw out png for each element in an easy way.

I have also created ChatBox to visualise the chat system and MusicBox class for music and sound controls, and I found the copyright free music and sound effects that suitable for our game style. During the development, I had chance to work with others and gave valuable experiences to work with other teammates and learnt from them along with their working attitude and coding style.

Team Project individual contributions assessment

Your name: Raymond Yiu Wai Chu Your team: B2

Team member	Contribution (%) (should sum to 100%)
Aidan Goodwin	**16.6%**
Atanas Harbaliev	**16.6%**
Christian Iliev	**16.6%**
Lucy Mills	**16.6%**
Raymond Yiu Wai Chu	**16.6%**
Samuel Tebbs	**16.6%**

Aidan Goodwin's Reflection

Reflection on teamwork:

Overall I feel that the team worked well together throughout the duration of the project. We met within the first few days of the project and continued to meet regularly after that (at least once a week). Every member of the team attended these meetings regularly. Some of these meetings were used to discuss progress and to plan development, whilst others were used to develop and debug code together.
The regular meetings, along with continual use of git from the start of the project, and an active facebook group meant that the team worked well together and were clearly focussed on the current project goals. Any questions I had could be answered quickly by the rest of the team and any misunderstandings that arose were quickly and amicably resolved and did not hinder progress.
We had to develop and debug many sections of code which required two or more members of the team working together to complete, and in these instances I feel that we worked well together and were able to successfully bounce ideas off each other.
In conclusion I feel that our team worked together effectively and I would happily work with them again in the future.

Reflection on individual contribution:

Since the start of development I have worked primarily on the Rendering and UI aspects of the project, working closely with Raymond and Chris. I feel that I made a significant contribution to these areas of the project, where as well as developing most of the rendering code, I also created most of the sprites used in the game.
The most prominent features that I developed were the code which draws the world, players pop-up menus and other effects to the screen whilst playing a game, the code which manages user input whilst playing a game, and the code which manages animations.
I also primarily worked on the interaction between the user interface and game logic, ensuring that the ui updated in real time with the back end of the game.
Another key feature that I developed was a search function which hides parts of the world from the players which the player cannot see (i.e. other rooms).
Towards latter parts of the project I also spent a significant amount of time helping to debug and add functionality across all other areas of development, working with Lucy and Sam to develop the networking code and game logic respectively, but also occasionally helping out with AI debugging. This gave me a better knowledge of the interaction between components than most of the other members of the team. I feel that this was an essential role to take on as it helped to ensure that all of the areas of the project worked well together and I was able to help focus the team on areas which needed the most work.
Overall I believe that i made a substantial contribution to the project in both written code and organisation and was a central member of the group.

Team Project individual contributions assessment

Your name: Aidan Goodwin Your team: B2

Team member (include your own name in this list)	Contribution (%) (should sum to 100%)
Aidan Goodwin	16.6%
Atanas Harbaliev	16.6%
Christian Iliev	16.6%
Lucy Mills	16.6%
Raymond Yiu Wai Chu	16.6%
Samuel Tebbs	16.6%

Christian Iliev's Reflection

Reflection on my individual contribution

In the weeks before the prototype demo I was involved in the creation of the Rules menu, in addition to writing the rules section. I also helped create our Connect and Options game state, and the Main Menu, which are central to the game. My primary focus was the design and delivery of the User Interface, in keeping with the principles of unobtrusiveness. Our designs constantly improved, we were all focused on providing a clean, uncluttered, effective and usable User Interface. As part of this commitment, I was tasked with formatting fixes throughout the project's duration, choosing fonts and object interaction with the mouse, which after our usability testing we decided to completely alter as it was not intuitive enough.

In Week 8 most of the work on the game UI was nearing completion, with three of us working on it, so we decided on a flexible team organisation, where I had the opportunity to create one of our minigames, which every player encounters multiple times during the gameplay when they use the Work action - Hangman. I did the design, found a wordlist of more than 2000 challenging words, and wrote a parser for it, so we could have a randomly generated word every time with less chance of repetition. Afterwards I worked on the underlying game logic, and the game's visual representation. There was a difficulty however in making the game sync with our integrated version, which was resolved thanks to Samuel, with whom I collaborated to rework the game logic component according to the minigame framework he had implemented.

Additionally, I created the UI usability testing strategy, carried out the tests themselves - gathering feedback from 5 players, which I then analysed (see Appendix). Last but not least, over the course of the last week I played a fundamental role in the production of the final report.

Overall, this project has taught me that team communication is key in resolving development and design issues. I also learnt that importance of testing with real users of the system, especially when it comes to design, as this provides certainty whilst saving time refactoring - something highly valuable in the end stages of the development process. I feel more confident in my coding abilities and I am thankful that I had the opportunity to work alongside such knowledgeable and highly-motivated teammates, who helped me learn and improve.

Reflection on teamwork

I can conclusively say that we were a very effective and determined team. Although having not been acquainted with my teammates before we set on our project, I immediately noticed we all shared a passion for developing a great game. Help debugging or creating new features was always on hand, issues were resolved quickly, thanks to us meeting regularly more than once a week for extended periods of time, in addition to our group chat, which allowed for swift communication of plans and problems alike. I believe one area of improvement would have been more frequent updates on what was accomplished already, shared with all teammates.

It took us very little time to begin functioning as a united, yet flexible whole, supportive of each other, the result of which I feel is undeniably successful - DevWars.

Team Project individual contributions assessment

Your name: Christian Iliev Your team: B2

Team member (include your own name in this list)	Contribution (%) (should sum to 100%)
Aidan Goodwin	16.6%
Atanas Harbaliev	16.6%
Christian Iliev	16.6%
Lucy Mills	16.6%
Raymond Yiu Wai Chu	16.6%
Samuel Tebbs	16.6%

Samuel Tebbs' Reflection

Reflection on teamwork:

At the start of the project we all met up to discuss ideas and agreed on the one that would become the final product. Right from the start, we set up the git repository, decided upon a branching model, agreed to meet at least once a week (sometimes twice) and created a group chat on Facebook through which we communicated throughout the 11 weeks. The delegation of roles was also very simple as we were all flexible in our preferences and quickly settled on a good role distribution. All group members attended the meetings diligently and regularly, which meant that there was always someone one hand to discuss ideas with and report bugs to.

The teamwork itself was very good and there were very few instances where we couldn't combine our efforts in fixing bugs, coming up with ideas and discussing improvements. This efficient and enjoyable teamwork was facilitated by good team-wide use of Git, packaging and use of the shared game logic library.

I feel that we all got along during the entire project and we became more and more seamless in our work together as the 11 weeks progressed, and as we got to know each other more.

In conclusion, I feel that we all worked very well together, fit as a cohesive unit and got the job done. I would like to work with them all again if the opportunity arises.

Reflection on my individual contribution

I was tasked with developing the game logic, which I knew would be a big task but I was up to the challenge. I first started creating the underlying representations of the game world (such as the players, world itself and level format) and then worked on the event system, which we used for interaction between the game's various subsystems. I did my best to make the systems I created as generic and extendable as possible, so that future modifications and additions would be as seamless, integratable and fault-free as possible. My priority was making a usable and efficient system that the other team members could utilise to full effect. I tried to consult the other members of the team to ensure that what I had implemented suited them as well as possible.

In addition to the game logic, I also helped debug some parts of the networking and AI, as well as assisting with any Git problems other team members had, although the game logic consumed the vast majority of my time on the project.

In conclusion, I believe that I made a significant contribution to the project, which was necessary as the game logic designer and developer.

Team Project individual contributions assessment

Your name: Samuel Tebbs Your team: B2

Team member (include your own name in this list)	Contribution (%) (should sum to 100%)
Aidan Goodwin	16.6%
Atanas Harbaliev	16.6%
Christian Iliev	16.6%
Lucy Mills	16.6%
Raymond Yiu Wai Chu	16.6%
Samuel Tebbs	16.6%

Atanas Harbaliev's Reflection

My responsibility, during the team project, was to develop the AI element of our game. After finishing the planning bit, I started writing the pathfinding part of the AI system. For it I used A* algorithm. The first few days, I did some research and then started writing my own class. After successfully implementing it, I had to adjust it, so it could work on the game's map. Also, a few tweaks were required in order to be able to integrate it with the Networking part of our project. For example, inner classes had to be made separate, the implementation of pairs was required. After finishing with pathfinding I started working on the logic behind the AI's behaviour. For this part, I created an Interface called Logic and then two classes implementing it – LogicEasy and LogicHard. Those two classes contain information about how and when the AI player should do actions like work, hack, refresh, etc. Implied by the names, the difference between those two classes is that LogicEasy has some silly implementations that make the AI player act sub-optimally. Also, depending on the mode, the AI player will move at different speed. Finally, I had to decide on a system that will control the AI and tell it when to do what. I decided to implement a Rule based system, as we needed a one that could be implemented in a few weeks, work efficiently, and accurately. Since this is the first time I had to implement such a system, I did spend a while studying it. My implementation of it is consists of Working memory for a player, set of rules, and updater object and an object that fires certain rules depending on the situation.

Reflecting on the work I did during the project, I must say I was working regularly on the project, did use the version control system every time I did fix/implement something new, was on every meeting our team had. As well as implementing the AI for our game, I did help with debugging and integration, partially helped with the documentation and both presentations.

To sum up, I believe we did good working as a team. Everyone was doing their best and worked hard towards the completion of the project. We could have done more meetings and used more software engineering techniques, but overall, it is my opinion that we did a good job collaborating with each other .

Team Project individual contributions assessment

Your name: Atanas Harbaliev Your team: B2

Team member (include your own name in this list)	Contribution (%) (should sum to 100%)
Aidan Goodwin	16.6%
Atanas Harbaliev	16.6%
Christian Iliev	16.6%
Lucy Mills	16.6%
Raymond Yiu Wai Chu	16.6%
Samuel Tebbs	16.6%

Testing

This section details the testing strategy employed and the principles abided by when testing DevWars.

Testing was done on a component basis, and where testing was done by JUnit, the names of the test classes are organised by development section e.g Networking, Game Logic etc. and listed under JUnit Test Classes in Appendix 3, attached to the end of this report.

Game Logic Testing Strategy:

1. Tested the chat system to ensure chat messages successfully reach the appropriate players
2. Tested the events system to ensure that all registered event listeners are called at the appropriate time and in the correct order (based on priority)
3. Created mini game tests to make sure that the mini games run smoothly and as intended
4. Created player action tests to ensure that the actions have the intended effects
5. I made a player status test to ensure that all functionality regarding player status (actions, effects, states) work properly
6. I implemented tests for the server and client synchronisation to ensure that the game state is properly synchronised across networked users
7. The utility classes were tested to make sure they do not cause the rest of the game logic to fail and that they fulfill their intended utility purpose.
8. The functionality of the world classes was tested to ensure that the game state was properly managed, added to and removed from.

AI Testing Strategy:

The AI element of DevWars consists of a path finding element, a logic one, which splits into easy and hard modes, and the third and final element - a rule based system. In order to achieve a working AI, all three components were tested both individually, as well as in conjunction with each other.

Firstly, the path finding algorithm was tested, which finds the closest, out of two or more, goal points and calculates the shortest path to it using the A* search algorithm. Since players in our game can only move north, south, east, and west, paths which have same final values like {north, north, east} and {north, east, north} are both valid, yet different. This was approached by finding all paths with an equal final value and comparing them to the output of the A* algorithm.

Secondly, we had the ability of the AI to work and refresh tested. Since there are two AI modes - hard and easy, which implement the same interface, but have different implementations, we wrote different JUnit tests for the two logics. The main idea behind both is to test whether the AI player is working and refreshing. This was approached by simulating the process of work for a certain amount of time to check if the AI is gaining progress and then simulating the event of the AI player going to the coffee machine to check if its fatigue was reset.

Thirdly, we tested the rule system. To do so, we simulated all possible conditions that might emerge while playing the game and tested how the working memory was being updated.

Finally, we ran all test multiple times to make sure that the AI was robust and doing everything it is supposed to do.

Networking Testing Strategy:

The network has had black box testing, grey box and white box testing mainly using JUnit testing.

For black and grey box testing we ran the 4 player game on one computer, 4 computer on the same network and 4 computers on different networks. Initially we ran the game as normal and it ran smoothly. We then tried a client disconnecting part way through the game and checking an AI player was added. When connecting clients we ensured that if two people entered the same name the network handled this by adding an extension to one of the players. Next we tried only two people connecting and ensuring a second AI player was added, this worked. Also, after a game is finished we connected a new 4 players to make sure the game restarted. Furthermore, we tested constantly pressing the drink actions to ensure that the network could handle the constant sending of object and didn't produce serialization issues like the network did before the message queue was added. The testing was carried out by the team member who wrote networking as well as the ones who didn't. We also got other people outside the team to play the game and told them to mess with it how they wanted to ensure the network didn't crash, this was successful. We tried all of these tests multiple times to ensure the network was robust and worked every time the game was run. We also carried out these test on different computers over local and non-local networks.

For white box testing we tested different paths through the code such as players with the same name being added, players leaving, and waiting for too long for players which results in an extra ai being added. In order to test this we added code breaks to the necessary classes and ran the code in the debugging mode in eclipse to ensure it was following the intended path and the correct output was being produced for the right reasons. We also tested a client connecting to the server using JUnit tests. The results of these can be seen in the appendix. However due to the implementation of the server we were unable to carry out any other JUnit tests however extensive testing has been performed in other ways.

UI Testing Strategy:

In order to meet our primary objective of delivering a clean, uncluttered, and — more importantly — effective user interface, usability tests were the main strategy employed. We recognised early on that is very unlikely that we will be able to design a good product without doing some kind of research and testing.

A key aspect of usability testing is to involve the users as much as possible. Our 5 test subjects who have not seen the game hitherto were asked 5 binary questions related to intuitiveness and ease of use whilst performing certain actions, which were also monitored by the member of the team tasked with conducting the tests.

We were looking for feedback that included the visuals and the way interactions were performed, excluding the game logic and the performance of the game. The test players offered their feedback and the UI team made changes according to the suggestions.

The following questions were asked:

1. Did you find navigating the main menu intuitive?

2. Is it clear what this screen represents at a first glance? *(Interviewer comment: related to the player status overview, shown by pressing the Tab key)*

3. Would you prefer controlling the player yourself, as opposed to automatic routing when you click on a given object?

4. Are the fonts chosen suitable for a game of this type?

5. Do you find interacting with objects in the game swift and easy?

In addition to that, further feedback was gathered in our weekly meetings by those members of our team who were not involved in the UI design and implementation.

A detailed summary of all feedback, including a graphical breakdown of the answers from the user interviews, and the actions undertaken in response thereto can be found in Appendix 7: UI Testing.

Appendices

Appendix 1: References

Appendix 2: Coding Standards

Appendix 3: List of JUnit Test Classes in the project

Appendix 4: AI Testing

Appendix 5: Game Logic Testing

Appendix 6: Networking Testing

Appendix 7: UI Testing

Appendix 8: UML Class Diagram for the Game Logic component

Appendix 9: UML Class Diagram for the User Interface component

Appendix 10: UML Class Diagram for the Networking component

Appendix 11: UML Class Diagram for the AI component

Appendix 12: UML Class Diagram for the supporting classes

Appendix 1: References

[1] http://www.ponggame.org
[2] http://gameprogrammingpatterns.com/game-loop.html
[3] http://www.scrumstudy.com/blog/what-is-crystal/
[4] http://wiki.c2.com/?ReflectionWorkshop

Appendix 2: Coding Standards

This document contains the coding conventions, based on Google's Java Style Guide and the Java Language Specifications, with adaptations by the B2 Team during the development of the DevWars game.

1. Source File Basics
    1.1. File Names
    The source file name consists of the case-sensitive name of the top-level class it contains (**of which there is exactly one**), plus the `.java` extension.
    1.2. Source file structure
    Each Java source file contains a **single** public class or interface.
        1.2.1. A source file consists of, in order:
            1.2.1.1. Beginning comments (license or copyright information where applicable)
            1.2.1.2. Package statements
            1.2.1.3. Import statements
                1.2.1.3.1. Import statements are **not** line-wrapped.
    1.3. Ordering of class contents
        1.3.1. When a class has multiple constructors, or multiple methods with the same name, these appear **sequentially**, **with no other code in between** (not even private members).

2. Formatting
    2.1. Braces
        2.1.1. Braces are used throughout
        Braces are used with `if, else, for` and `while` statements.
            2.1.1.1. Exception: **if the body contains a single statement, braces are not required.**
        2.1.2. Empty code blocks may be concise
        An empty block may be closed immediately after it is opened, with no characters or line break in between.
        Example: `void functionCopy() { }`
        Note: `void functionCopy() { newline }` is also acceptable.
    2.2. Column limit of a 100 (Eclipse default)
    2.3. Line wrapping
    Using the Eclipse default settings (Ctrl/Cmd + Shift + F)
    2.4. Whitespace
        2.4.1. A single blank line whitespace appears:
            2.4.1.1. Between consecutive members or initializers of a class: fields, constructors, methods, nested classes, static initializers, and instance initializers.
            2.4.1.2. Between statements
            2.4.1.3. After the end of the import statements in 1.2.1.3.
        2.4.2. Horizontal whitespace
            2.4.2.1. Separating any reserved word, such as `if, for` or `catch`, from an open parenthesis (() that follows it on that line
            2.4.2.2. On both sides of any binary or ternary operator.
            Exception: dot separator(.) and two colons(::)

2.4.2.3. On both sides of the double slash (//) that begins an end-of-line comment.
Example: `int x = 1280; // The initial value of the screen width`

2.5. Variable declarations

2.5.1. **One variable per declaration**
Every variable declaration (field or local) declares only one variable: **declarations such as `int a, b;` are not used.**

2.6. Annotations

2.6.1. Each annotation is listed on a line of its own.

3. Naming

3.1. All identifiers use **only** ASCII letters and digits

3.1.1. Package names

3.1.1.1. Package names are all **lowercase**, with consecutive words concatenated together without underscores.
Example: `game.ui.components`

3.1.2. Class names
Class names are written in **Upper Camel Case** (e.g `ServerListener`)

3.1.3. Method names
Method names are written in **lowerCamelCase** (e.g `getConnection`)

3.1.4. Constant names
Constants are static final fields. Constant names use **CONSTANT_CASE**: all uppercase letters, words separated by underscores

3.1.5. Parameter names, Local variable names and field names are written in **lowerCamelCase**.

4. Javadoc

4.1. Formatting

4.1.1. Both multiline and single-line Javadoc blocks are accepted.

4.1.2. Block tags (e.g `@param, @return`) are never left empty.

4.1.2.1. Exception: @throws

4.1.3. **All** Javadoc blocks should begin with a short description sentence(s).
Example: `/* Returns the world to be rendered */`

4.2. Usage

4.2.1. Javadoc is present for every class and method with the following exceptions

4.2.1.1. Self-explanatory methods
Examples: `String getName(), int getIndex`

4.2.1.2. Methods with `@Overrides`

Appendix 3: List of JUnit Test Classes in the project

All test classes can be found under the test/game package.

AI:
PathFindingTestEasy
PathFindingTestEasy
LogicEasyTest
LogicHardTest
FireRulesTest

Networking:
NetworkTesting

Game Logic and Game Core:
ChatTest
ChatMessageTest
EventsTest
InputTypeTest
MiniGameTest
MiniGameHangmanTest
MiniGamePongTest
PlayerActionTest
PlayerEffectCoffeeBuzzTest
PlayerEffectOnFireTest
PlayerStatusTest
PlayerTest
ClientSyncTest
ServerSyncTest
UpdaterTest
CoordinateTest
DataHolderTest
TileTypeTest
LocationTest
WorldTest
PairTest
SetsTest
TestUtil

Appendix 4: AI Testing

The results from running the path finding tests were positive. One of the possible paths was always correct, which proves the correctness of the implementation of A* algorithm. Once we knew which path the algorithm outputs, we removed the failing cases.

When it comes to the hard and easy logic tests, both passed all tests with conclusive results, showing that everything is working properly.

Finally, the ruled based system's output was positive, as well. Every time the memory was holding the correct values for the correct player.

Appendix 5: Game Logic Testing

Below is a report of the game logic coverage testing:

[all classes]

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	94.5% (103/ 109)	80% (364/ 455)	81.1% (987/ 1217)

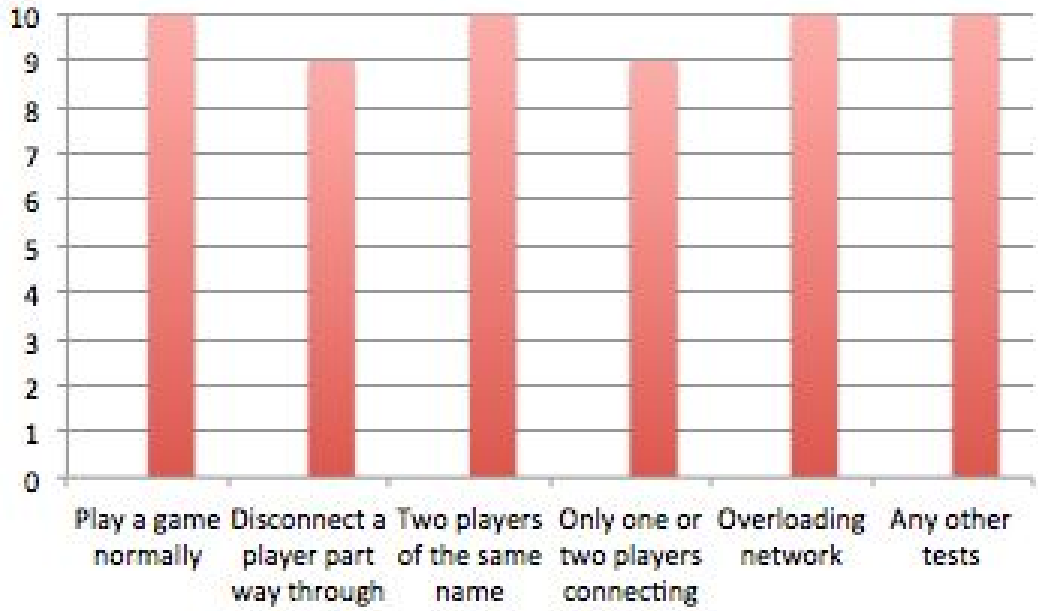
Coverage Breakdown

Package	Class, %	Method, %	Line, %
game.core	100% (1/ 1)	100% (1/ 1)	100% (2/ 2)
game.core.chat	100% (2/ 2)	85.7% (12/ 14)	82.9% (29/ 35)
game.core.event	60% (6/ 10)	70.6% (12/ 17)	70.3% (26/ 37)
game.core.event.chat	100% (3/ 3)	66.7% (6/ 9)	70.6% (12/ 17)
game.core.event.minigame	100% (4/ 4)	83.3% (5/ 6)	94.4% (17/ 18)
game.core.event.player	91.7% (11/ 12)	91.7% (11/ 12)	89.2% (33/ 37)
game.core.event.player.action	100% (3/ 3)	100% (3/ 3)	100% (7/ 7)
game.core.event.player.effect	75% (3/ 4)	75% (3/ 4)	63.6% (7/ 11)
game.core.event.tile	100% (3/ 3)	100% (3/ 3)	100% (13/ 13)
game.core.input	100% (10/ 10)	100% (14/ 14)	100% (22/ 22)
game.core.minigame	100% (6/ 6)	100% (38/ 38)	91.5% (150/ 164)
game.core.player	100% (3/ 3)	91.1% (51/ 56)	94.5% (156/ 165)
game.core.player.action	100% (9/ 9)	78.4% (40/ 51)	79.3% (88/ 111)
game.core.player.effect	100% (4/ 4)	80% (12/ 15)	85.7% (30/ 35)
game.core.player.state	100% (3/ 3)	78.6% (11/ 14)	78.6% (11/ 14)
game.core.sync	100% (5/ 5)	83.7% (36/ 43)	82.7% (115/ 139)
game.core.util	100% (2/ 2)	88.2% (15/ 17)	86.1% (31/ 36)
game.core.world	100% (3/ 3)	83% (44/ 53)	82.9% (121/ 146)
game.core.world.tile	100% (3/ 3)	70% (7/ 10)	54.3% (19/ 35)
game.core.world.tile.metadata	100% (2/ 2)	100% (3/ 3)	100% (8/ 8)
game.core.world.tile.type	100% (14/ 14)	48.3% (29/ 60)	51.4% (73/ 142)
game.util	100% (3/ 3)	66.7% (8/ 12)	73.9% (17/ 23)

All 131 implemented tests passed.

Appendix 6: Networking Testing

The network was tested in the NetworkTesting class. The results from connecting a client to a server were positive. The JUnit test was ran multiple times and passed every time.
The results from getting team members and peers to test the network can be seen below. The two 9's were from an occasional concurrency issue in Game Logic.



Appendix 7: UI Testing

Section 1: Breakdown of question responses (questions on page xx)

Section 2: Detailed feedback from UI testing and actions undertaken

Main menu

1. Feedback from testing showed that navigating with the mouse in order to access menu elements, especially on a screen with high resolution, was not effortless.
    1.1. Actions taken:
        - The main menu can now be navigated with the arrow keys only.
        - The order of player name and server address fields was swapped.

In-game

2. **Player Status Screen**
    2.1. Testing showed that it was not obvious what each bar represented, despite them being colour coded.
        2.1.1. Action taken:
            - Discrete labels were added to both the Progress and the Fatigue bars.
3. **Movement**
    3.1. Moving around in the game was accomplished by clicking on the desired object, after which the player was moved automatically by the system. Not only did this add unneeded computing overhead, but it took away from the user experience by preventing interaction with the game.

3.1.1. Action taken:

- The play now has to navigate the map themselves using the WASD keys.

4. **Fonts**

4.1. A default font (Arial), which is crisp and too contemporary, was used in the game. It did not fit the overall look and feel we are striving for, which is supposed to be playful and old-fashioned.

4.1.1. Action taken:

- A new font was self-created by using Photoshop that fits the style of the game. An added benefit is extensibility - this font is not composed of all possible words, but uses single letters and builds strings according to what is needed.

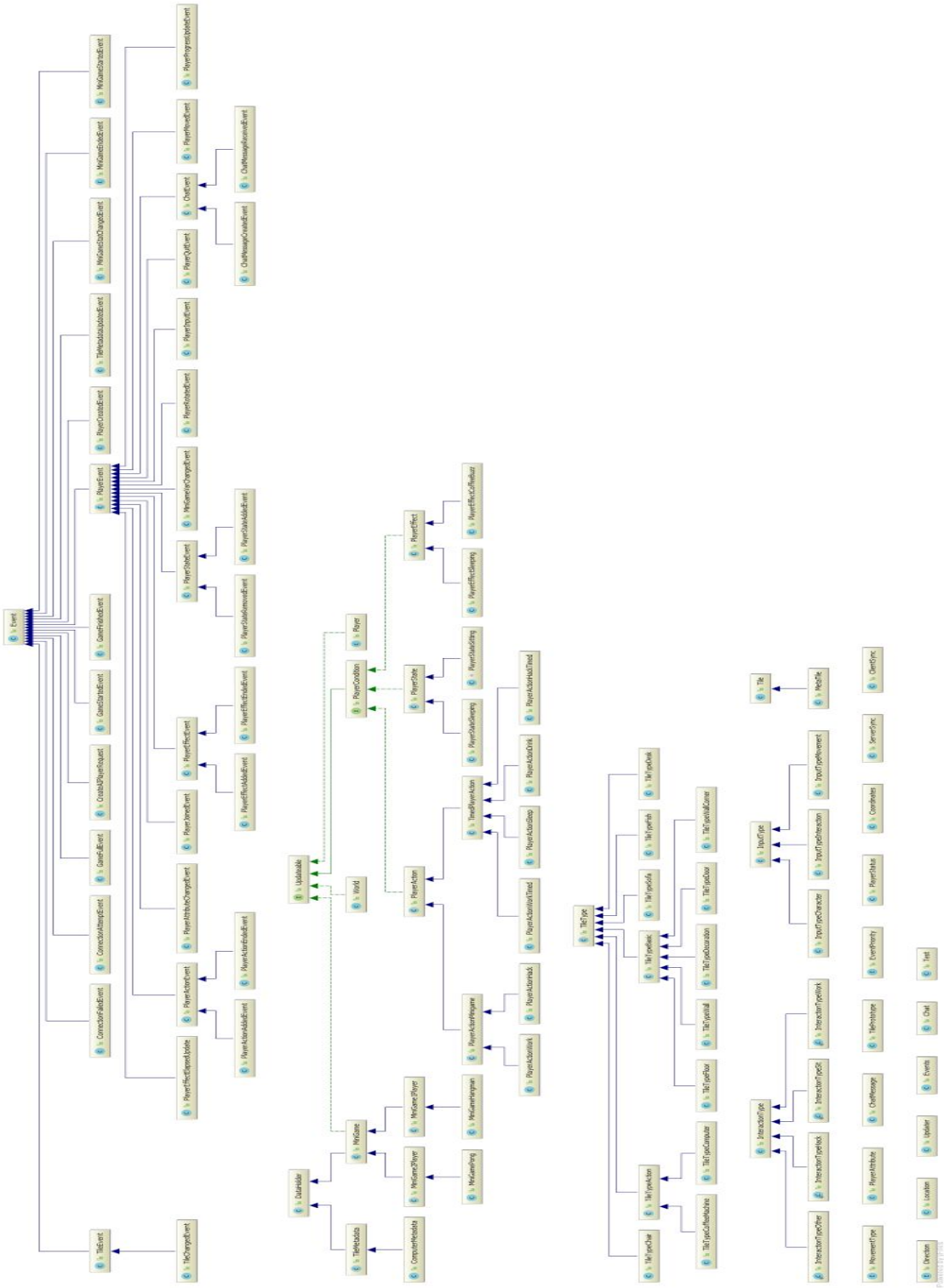
5. **Interaction menus navigation**

5.1. The interaction with objects in the game used the mouse scroller. Testing showed that while this is useful with a physical mouse present, most users and us, the developers, used a laptop trackpad, which made it impossible to track how many scrolls were performed, leading to overscrolling and inaccuracies.

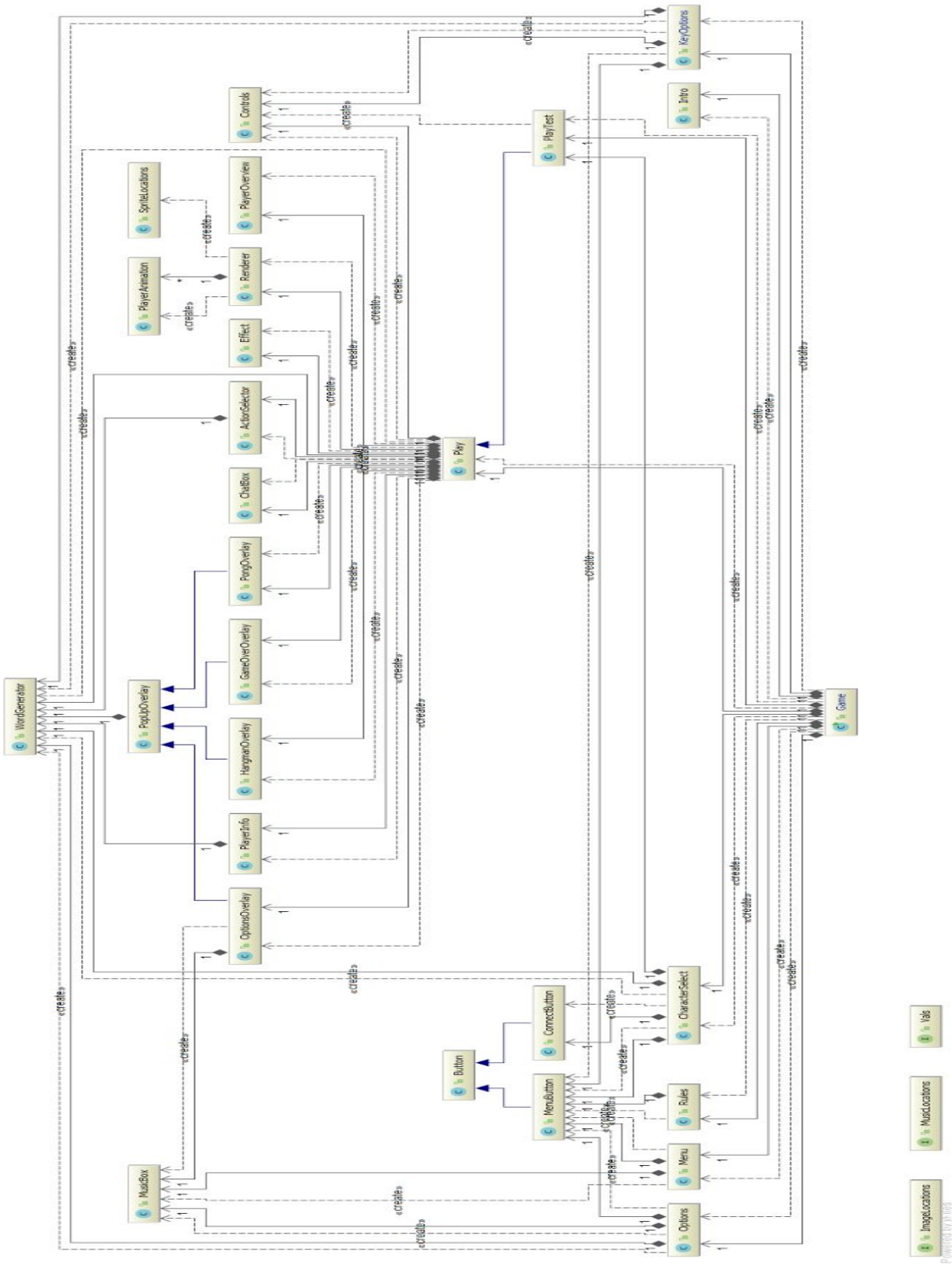
5.1.1. Action taken:

- This was changed to the arrow keys, as they are ubiquitous on all types of computers, in addition to being intuitive i.e Up and Down move through the list one step at a time in the specified direction, whereas Left moves back.

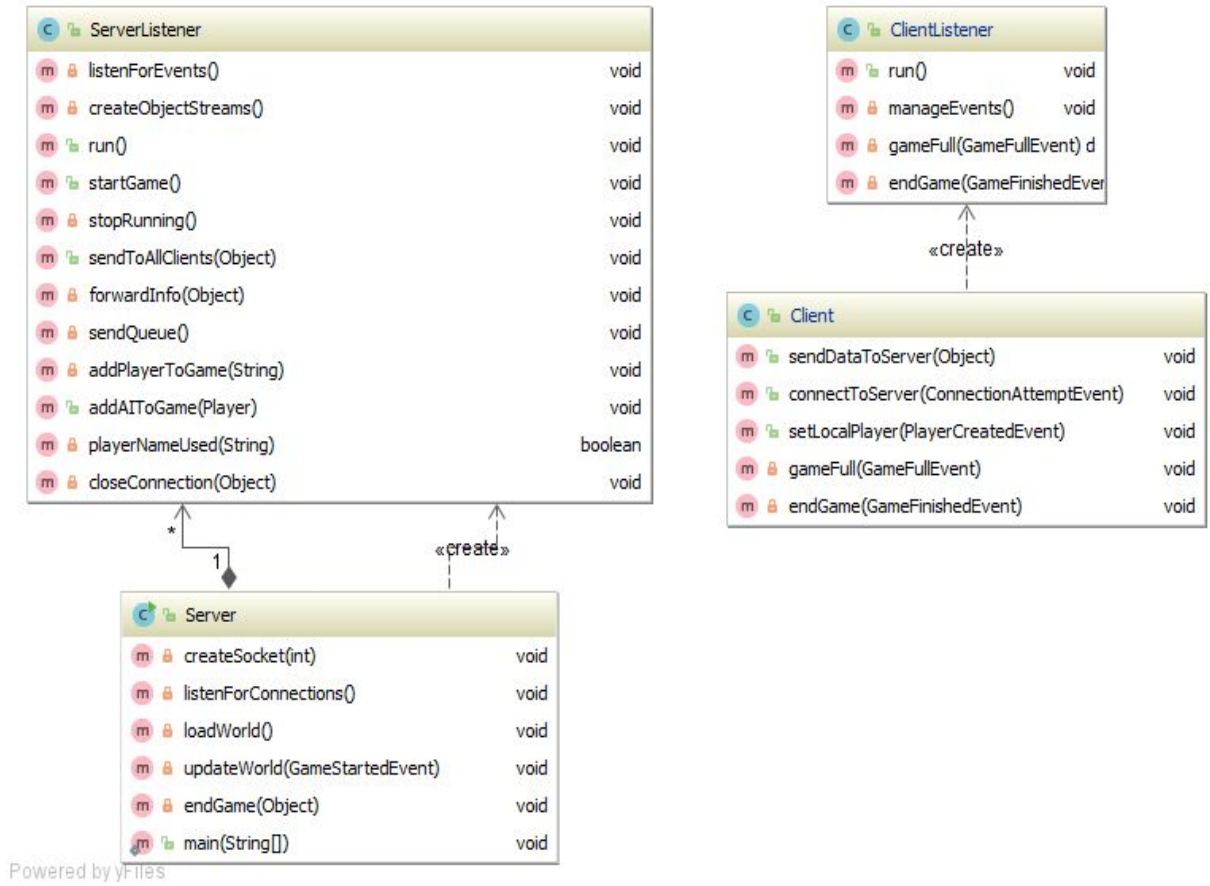
Appendix 8: UML Class diagram for the Game Logic component



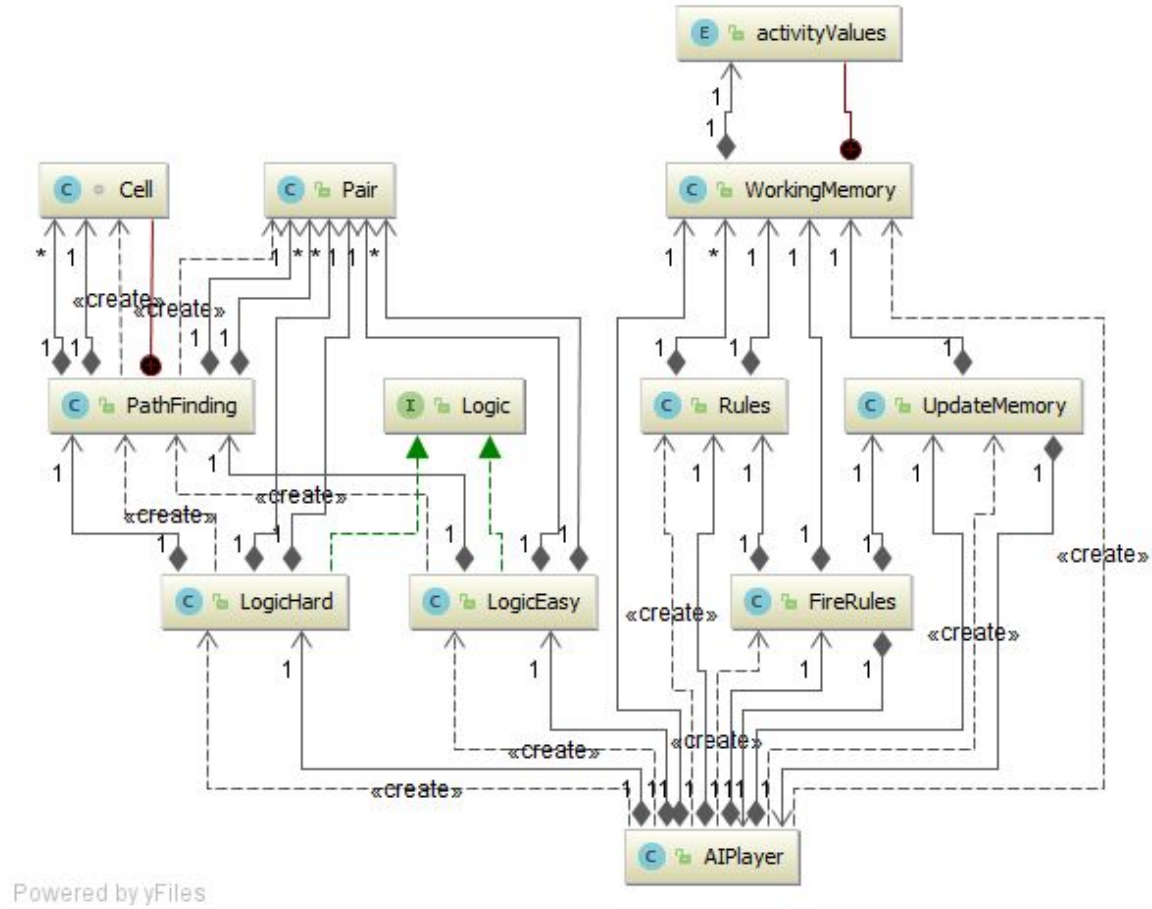
Appendix 9: UML Class diagram for the User Interface component



Appendix 10: UML Class diagram for the Networking component



Appendix 11: UML Class diagram for the AI component



Appendix 12: UML Class diagram for the supporting classes (Util)