How Rational Are You?

Third Year Project Report 2013

University of Manchester
School of Computer Science

Written by Shahab Rostami
Supervised by Dr. Joshua Knowles

Abstract

How Rational Are You?

Author: Shahab Rostami

Supervisor: Dr. Joshua D. Knowles

Although there is a growing interest in books and published materials relating to rationality, there is an absence of software applications that provide an evaluation of a user's rational behaviour. This project aims to address this issue by creating a two-dimensional online multiplayer game which enables users to assess their rationality in various situations.

The game implements a client-server network enabling simultaneous multiplayer game sessions with online capability. This project uses a board game approach providing a unique medium for users to participate in questions, play mini-games and receive feedback. This has been fundamental in providing a means of measuring user's rationality in an enjoyable and competitive environment. Testing has been conducted throughout this project ensuring that the final build of the game is bug-free, balanced and meets user requirements.

This report will detail how the above has been achieved, the approach taken and the results produced, demonstrating that all the project objectives have been successfully met.

Acknowledgements

I wish to thank the following people for their help and guidance throughout this project:

- Dr. Joshua Knowles for his supervision in my final academic year, providing insightful advice and guidance.
- Matthias Mann for enabling the integration of the Slick2D engine into his Themable Widget Library.
- Nathan Sweet for taking the time to support the use of his Kryonet library in this project.
- The vast amount of testers and the 100 respondents of the audience survey, contributing valuable results that have helped shape this project.
- The open-source community contributing to the numerous technologies used throughout this project.
- My girlfriend, friends and family for their unconditional support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Glossary

2D – Two-Dimensional

3D – Three-Dimensional

AI – Artificial Intelligence

API – Application Programming Interface

BSD - Berkeley Software Distribution

CSS – Cascading Style Sheets

CTP – Community Technology Preview

FPS – Frames per second

GPL – General Public License

GUI – Graphical User Interface

HTML – HyperText Markup Language

IDE – Integrated Development Environment

JNLP – Java Network Language Protocol

JVM – Java Virtual Machine

LWJGL – Lightweight Java Game Library

NAT – Network Address Translation

NIO – New I/O

OpenAL – Open Audio Library

OpenCL – Open Computing Language

OpenGL –Open Graphics Library

P2P – Peer-to-Peer

RMI – Remote Method Invocation

SDL – Simple DirectMedia Layer

TCP – Transmission Control Protocol

UI – User Interface

UDP – User Datagram Protocol

TWL – Themable Widget Library

XHTML – Extensible HyperText Markup Language

XML – Extensible Markup Language

1 INTRODUCTION

This report will discuss the software development life-cycle for a cross-platform, online multiplayer game titled 'How Rational Are You?'. A review of the information required to follow this report will be provided, followed by details of the research, design and implementation conducted during the project development cycle.

This introductory section discusses the context of the project, explaining the problem that it hopes to solve, the current solutions in place and a detailed description of the proposed solution. Finally, an overview of the report will be provided, outlining the subsequent chapters.

1.1 Background of the Problem

Acting rationally or irrationally is a part of everyday life. Our actions play into our personality traits and what one person believes to be a rational decision could in fact be the complete opposite. Consider the following problem [1]:

> **A bat and ball cost £1.10.**
> **The bat costs £1 more than the ball.**
> **How much does the ball cost?**

This simple puzzle encourages an answer that is "*intuitive, appealing and incorrect*" [1]. This answer is 10 pence, which would give a total cost of £1.20. Daniel Kahneman, Professor of Psychology Emeritus at Princeton University, presented this problem to students in many universities across America and received significant results with over 80% of students giving this incorrect answer [1]. Kahneman studied these results; he determined that people who gave this incorrect response overlooked an obvious social cue and did not query the fact that such a simple puzzle was included in the questionnaire. It can be argued that such behaviour is irrational. Keith Stanovich, the Canada Research Chair of Applied Cognitive Science at the University of Toronto, explains that *"rationality should be distinguished from intelligence"*, in his interpretation, *"superficial or lazy thinking is a flaw in the reflective mind, a failure of rationality"* [2]. An audience survey, conducted as part of this project, presented the problem to 100 students, with over 50% of respondents answering incorrectly (see **Appendix A**).

This problem and many others have received attention for obtaining results that exhibit irrational behaviour. Many people have shown interest in learning about rationality, this has been demonstrated by the popularity of related questionnaires and the success achieved by various books, especially those of Kahneman, discussed in **Section 2.1**.

For this reason, this project aims to provide an environment for users to evaluate their behaviour in various situations – such as the problem above – and review whether their actions are as rational as they believe. The platform for achieving this will be a software application, which will be discussed further in **Section 1.3**.

1.2 Current Solutions

There is currently no standard form of assessment for rationality. However, a number of solutions attempt to evaluate an individual's rational behaviour through questionnaires or studies in psychology and economics.

Questionnaires typically involve some form of logical reasoning, situational analysis or probability. The ability to discern whether these questionnaires are provoking rational behaviour is difficult and their validity is often disputed. The audience survey (**Appendix A**) asked respondents whether they feel that these questionnaires are accurate, with only 2% indicating agreement. The most popular questionnaires will be compared and analysed later in the literature survey (**Section 2.1**).

A number of experimental studies in psychology and economics explore rational choice wherein an individual is required to determine the best action to take, given various scenarios. The behaviour demonstrated by the individual in these studies will play a vital role in evaluating their rationality. A number of these studies will be integrated into the project to provide different environments for users to interact and demonstrate rational behaviour; this is discussed further in **Section 2.1**.

There is an absence of software application solutions; this further supports the need for the proposed solution introduced in the next section.

1.3 Proposed Solution

The section begins by discussing the project's definition of rationality. The proposed solution will then be detailed followed by the aims and objectives of the project.

1.3.1 Defining Rationality

It is important to specify what is meant by being "rational" in order to assess rationality itself. It is a term that is difficult to describe; it has various meanings that can be obscure and often confusing. This project aims to evaluate the user's rational behaviour through rational choice theory. To the majority, "rational" is a term that describes an act of being "sensible", "sane" or acting "in a thoughtful clear-headed manner" [3]. Rational choice theory suggests that rationality is an individual's ability to maximise personal advantage by balancing costs against benefits [4]. The proposed solution will use this definition as the basis for evaluating an individual's rationality.

1.3.2 Solution Overview

The project aims to create a 2D (two-dimensional) two-player game that brings together existing questions and studies related to rationality from reliable sources. A unique board game approach will be utilised as the platform for these methods.

Players will take turns in rolling dice, allowing them to navigate the board. This will then determine whether players will answer a multiple-choice question or play a mini-game. These mini-games are the studies adapted into games for a two-player environment. Users are able to win points by answering questions correctly and gamble these points during mini-games. The player with the most points at the end of the game is the winner.

Feedback is an important factor for any application that aims to evaluate the user's behaviour. After a period of 15 minutes has elapsed the game will end and users will be given a rank indicating the level of rational behaviour they have demonstrated. To provide useful and informative feedback, detailed statistics calculated from the users actions throughout the game will be provided at the end of their session. Questions are detailed on their relation to rationality and how the correct answer demonstrates rational behaviour. This allows players to view precise information on each question they have answered.

Network capability is required in order to enable multiplayer support. Using a client-server model, players will be able to host and join game sessions. The network will also support numerous instances of the game; this means that many players can play the game simultaneously.

An important aspect of this project is for the game to be fair and meet user requirements. By conducting continuous acceptance, balance and end-user testing the project will aim to ensure users enjoy a consistent and comfortable experience whilst playing the game.

1.3.3 Project Objectives

The overall aim of the project is:

To *create a network-capable game that allows multiple, simultaneous pairs of players to assess their rationality in a competitive, reliable and enjoyable environment.*

Based on the aim of the project, several key objectives have been identified:

#	Objective Definition
1	To investigate existing solutions and compile a list of mini-games and questions to be used in the game.
2	To create a complete, robust and bug-free game using an iterative model approach to development and testing.
3	To implement client-server architecture that allows synchronisation and quality service that can handle various network sequences successfully.
4	To develop a scalable game engine that will allow questions and their corresponding feedback information to be softcoded, ensuring that this information can be added and manipulated with ease.

Table 1.1: Listing of project objectives.

1.4 Report Overview

The report is divided into 7 chapters including this introduction:

Chapter 2 provides a literature survey; this includes research into related existing work that has contributed to the decisions made as part of this project. Related technologies and methodologies are discussed in this section.

Chapter 3 describes the main design aspects of the project. The requirements analysis and the design choices made for the game's main functionality will be detailed. The development process used will also be discussed. The chapter will reference game design principles to support certain decisions.

Chapter 4 describes the implementation of the project. The chapter will discuss the selected technologies for the project. The implementation of important modules in the game will then be detailed; these are either vital to the functionality of the project or particularly difficult to develop. Finally, the problems encountered during this phase will be explained.

Chapter 5 provides the results of the project. There is a step-by-step walkthrough of the game, listing screenshots and an overview of the functionality displayed.

Chapter 6 details the testing and evaluation carried out throughout the project. This includes bug-reporting, test scripts, balance testing and end-user testing. The tests are analysed and discussed, relating back to the project objectives.

Chapter 7 concludes the report by summarising the themes of the project. The achievements obtained and opportunities for future development will be discussed.

2 BACKGROUND LITERATURE SURVEY

This chapter discusses the research conducted into the current solutions mentioned in **Section 1.2** and the technologies that are available to meet the proposed solution detailed in **Section 1.3.3**. The network protocols and models appropriate for the project will then be analysed. The chapter will conclude with a summary of what was learnt from the survey.

2.1 Current Solutions Research

As briefly mentioned in **Section 1.2**, there is an absence of software applications that attempt to evaluate rational behaviour. However it is still important to survey the current solutions that exist and the approaches they have taken as this will contribute to the improvement of the proposed solution.

This section begins by comparing the current popular rationality questionnaires, going on to discuss the unique solution shown by the game "ir/Rational redux" and finally summarising the studies and questions researched for the project.

2.1.1 Questionnaires

In this section I will briefly compare the most popular rationality questionnaires based on the number of views, comments and shares in social networking.

Source	Description	Question Sources	Number of Questions
Big Think [5]	Big Think is a popular website that acts as a knowledge forum. They bring together news stories, write articles and present them for discussion for users.	Daniel Kahneman, Amos Tversky, Peter Watson	3
HelloQuizzy [6]	HelloQuizzy is a website dedicated to the creation and sharing of quizzes. Users can create their own questionnaires on any topic and share with other users.	Not Specified	8
University of Toronto Magazine [7]	University of Toronto Magazine is the official magazine of the University of Toronto, sent quarterly to more than 300,000 alumni and friends of the university.	Daniel Kahneman	5
Vanity Fair Magazine [8]	Vanity Fair is a magazine of pop culture, fashion and current affairs in America. There have been editions for four European countries as well as the U.S.	Daniel Kahneman	7

Table 2.1: Comparison of questionnaires.

Big Think presents a small quiz with 2 questions adapted from the research studies of Daniel Kahneman & Amos Tversky and a single question from Peter Watson. All of these questions have been held in high regard with each covering a different aspect of rationality: "loss aversion", "deductive reasoning" and "conjunction fallacy". Therefore, the questions used in this questionnaire are from reliable sources that have been previously used to evaluate a person's rational thinking.

The presentation of the questionnaire itself is a simple discussion board post; there is no user interface for an individual to input their responses but simply displays the questions and choices available with the answers following at the end with some discussion. The answers are accompanied by useful feedback and an explanation of the question along with its relation to rationality.

The questionnaires from **Vanity Fair Magazine** and **University of Toronto Magazine** are very similar, with the differences mainly being the number of questions involved. However the **Vanity Fair** quiz has the feedback hidden, requiring users to click a link to expand a section under each question. This stops users from accidentally glancing at the answers, which may occur in the other questionnaires. Overall the questions and their appended feedback provide a very useful evaluation of rationality; it has been received in high regard by its users as illustrated by the comments that have been posted [7] [8].

The questionnaire from **HelloQuizzy** is a publicly-shared quiz submitted by one of its users. Although it is user-submitted it is ranked as the most popular rationality quiz on the website. However it remains that the validity of the questionnaire is debatable as the questions themselves are also created by the user with no scientific backing or relation to previous psychological or behavioural economic studies. The user may also have no educational background in the field. In comparison to the Big Think quiz, this means that the questions are not from reliable sources and therefore may not be directly related to evaluating rationality.

Aside from this, HelloQuizzy is specifically designed for the creation and sharing of quizzes, providing a user friendly interface for both answering and receiving feedback on questions. Nevertheless, the content of the feedback received is another drawback. Users are given their result as a percentage accompanied by a single paragraph describing their evaluated level of rationality and a brief description of how the questions are related to assessing rationality - there is very little detail on the individual questions and their answers. The feedback also includes a comparison of your result against all other participants in the form of a line chart with the calculated average. Due to HelloQuizzy's user interface, the questionnaire is simple and easy to use but the questions and feedback are lacking in terms of content and validity.

To conclude, these questionnaires all claim to measure an individual's rational behaviour by providing questions taken, or adapted, from related publications. However due to these being presented through online articles, posts or even magazines the available user interaction is very limited, ultimately presenting the questions in simple text form, with very little feedback.

2.1.2 Ir/rational Redux

Ir/rational Redux [9] is a 2D browser-based graphical puzzle adventure game released in July 2012 and created using Macromedia Flash. The story is revealed through a first-person view animation with the player acting as the protagonist. Its gameplay is based around a unique language-based argument system and as the player progresses through the story, they are met with a series of puzzles involving propositional logic.

As the story unravels, the player must choose the correct statement that completes the argument given correctly using drop-down menus that list a number of choices available, as demonstrated in **Figure 2.1** below.

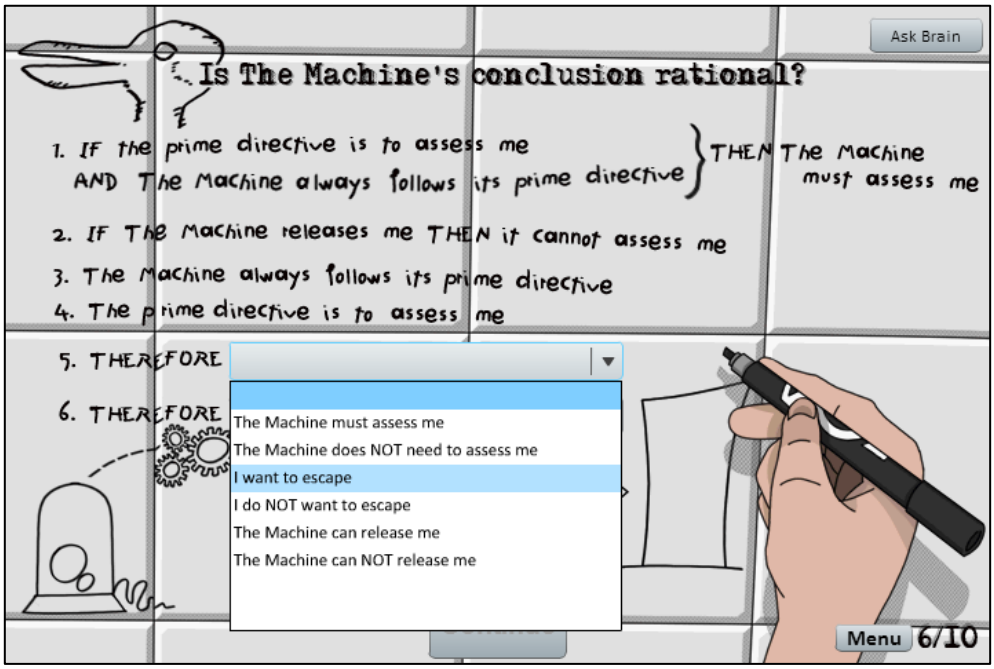


Figure 2.1: Screen capture of ir/rational Redux interactive gameplay.

The game presents a very interesting method of engaging a player through challenging and inventive gameplay, requiring the formulation of logical proofs to progress through the game. The use of animation, music and sound provides a rich and pleasant experience whilst educating the player on the topic of propositional logic [10].

The user interface is straight-forward and easy to use with no complex controls – a simple point and click game. However, a drawback to this game is the minimal amount of feedback given. After the game is completed the story ends and the credits roll, the player is given no information related to their behaviour during the game.

The public opinion of this game through comments and reviews rewards the game in its challenging gameplay and unique animation. However the game's validity is often questioned, with many players disagreeing with some of the propositional logic involved [10].

Overall, ir/rational Redux successfully demonstrates how a player can be captivated and educated through a 2D game.

2.1.3 Rationality Studies and Questions

An objective for this project, stated in **Section 1.3.3**, is to investigate and compile a list of previously published studies and questions that measure rational behaviour from reliable and valid sources. This section describes the studies chosen and the sources used to gather questions.

The studies below have been thoroughly analysed in game theory and were selected for this project:

- Prisoner's Dilemma,
- Trust Game,
- Sealed Bidding Auctions,
- Ultimatum Game.

Prisoner's dilemma [11] is a game where two players must choose to either cooperate or betray with the absence of knowledge of each other's decision. There are 3 different outcomes of this game.

- If both players cooperate, they each receive a reasonable reward.
- If one player betrays the other, the player who betrays receives a greater reward with the other receiving a lesser reward.
- If both players betray, they each receive the lesser reward.

This is a prime example in game theory where individuals might not choose to cooperate, even if it seems beneficial to do so.

Trust game [12] requires players to collaborate in order to increase a reward by a factor specified. For example, an initial endowment of £10 is given to Alice. Alice is told she can transfer any share of this amount to her partner, Bob, and keep the remainder. The amount transferred will be tripled and given to Bob, who can then return any share back to Alice. If Alice transfers a large amount, she is trusting, and if Bob returns a large amount, he is trustworthy.

Sealed Bidding Auction [13] requires players to bid on an item with the absence of knowledge in each other's decision. The players must place their bid in a sealed envelope and simultaneously hand them to the auctioneer. The auctioneer then compares the values and the player with the highest bid wins, paying the amount they have bid. For this game, the adaption in the context of game theory involves having both rewards and bids under the same currency. For example, the auction would be for a £10 reward, where players must bid in pounds.

Ultimatum game [14] requires both players to interact to decide how an amount is divided amongst them. For example, two players are shown £10. One of the players, called the "proposer", is instructed to offer a value between £1 and £10 to the second player, who is called the "responder". The proposer can make only one offer and the responder can either accept or reject this offer. If the responder accepts the offer, the money is shared accordingly. If the responder rejects the offer, both players receive nothing.

Questions: An initial list of 25 questions has been obtained from various sources, the majority from online publications and books based on behavioural economics, game theory or rationality. The following books have been used:

- *"Thinking, fast and slow"* by Daniel Kahneman [15].
- *"Judgement under Uncertainty: heuristics and biases"* by Daniel Kahneman [16].
- *"The Bounds of Reason: Game Theory and the Unification of the Behavioural Sciences"* by Herbert Gintis [17].
- *"Forever Undecided: A Puzzle Guide to Godel"* by Raymond M. Smullyan [18].
- *"Equilibrium and Rationality: Game Theory Revised by Decision Rules"* by Paul Weirich [19].

Each question has been assessed and compiled with the following:

- A number of choices, including the correct answer.
- A level of difficulty: easy, medium or hard.
- A document with detailed feedback of the question.

These games and questions provide a reasonable amount of content to evaluate the user's rationality whilst providing replay-ability.

2.2 2D Game Engines and Software Libraries

This section introduces game engines with a brief summary of their purpose. A summary of popular game engines that are appropriate for this project is provided, detailing the software libraries available for networking and graphical interfaces. Finally, a summary of the game engines is presented.

Game engines offer reusable components that deliver a system for the creation and development of games. Game engines abstract the details that are common in video game development, such as rendering, physics and user input. For this reason, they are typically considered as middleware. Games can be completed reliably and efficiently as they provide a framework for developers with essential functionality included, such as graphics, a physics engine, sound, animation, AI (artificial intelligence) and networking. This allows developers, from artists to programmers, to focus on the unique content of their game. Cross-platform capability may also be supported, allowing the game to be run on various platforms.

Name	Primary Programming Language	License
Microsoft XNA	C#	Microsoft EULA 2.0
Pygame	Python	LGPL
Slick2D	Java	BSD

Table 2.2: List of popular 2D game engines and libraries.

Table 2.2 lists the most popular 2D game engines in game development that are appropriate for this project, based on the level of activity and size of their communities. The central component typically provided by these libraries and engines is the 'game loop'. This acts as a default execution cycle imposed upon the user when developing a game.

The standard cycle is as follows:

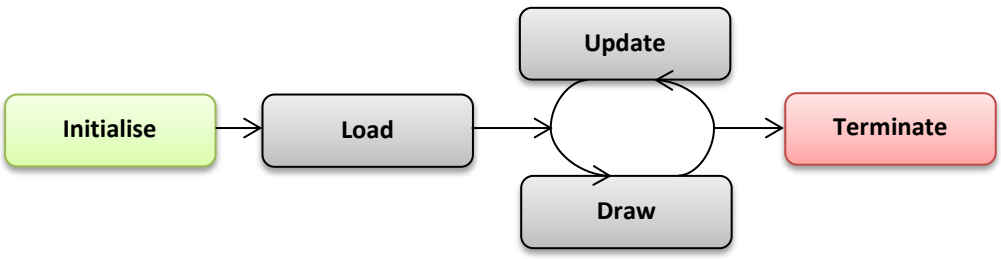


Figure 2.2: Standard game loop cycle.

1. First, the ***Initialise*** method is called; this handles standard operations such as memory allocation, resource acquisition and so on.
2. After this, the ***Load*** method is called, this prepares the game to be run following the initialisation method, as the name suggests.
3. The game then loops through both the ***Update*** and ***Draw*** methods until the game is terminated. The ***Update*** method handles all the game logic and the ***Draw*** method handles what is actually drawn to the screen.

Each game engine will now be analysed with its advantages and disadvantages followed by a discussion on its available network and GUI (Graphical User Interface) libraries.

2.2.1 Microsoft XNA Game Engine

XNA [20] is a toolset created by Microsoft for cross-platform game development between their Windows and Xbox related distributions. It was announced in 2004 at the Game Developers Conference in San Jose, California. On March 14th, 2006, a CTP (Community-Technology Preview) of the XNA Build was released. The XNA Framework is based on the .NET Framework 2.0, including useful libraries and APIs (Application Programming Interface) specific to common game-related tasks, encouraging code reuse across target platforms [21]. The XNA framework supports games written in any .NET compliant language, however C# is the only language officially supported.

A significant advantage of XNA is that its Game Studio is an extension for the Microsoft Visual Studio IDE, allowing straightforward development, debugging and deployment. The Content Pipeline that is provided handles any media resources included in the game. This automatically converts the resource to the correct format for the Windows platform it is being run on. XNA is also easy to use whilst being extensively well-documented with plenty of tutorials and example code available due to its large active community. The main drawback is that it only runs on the .NET framework, meaning the games developed can only be run on Windows platforms.

XNA Network Library:

With XNA 2.0 released in 2008, the framework included a high-level networking API to create multiplayer online games [22]. This includes core functionality to facilitate networking for a game and the developer is left to choose what type of model to use (client-server or peer-to-peer), sending game data over the network and dealing with network latency (prediction and interpolation). Developers would frequently have to worry about packet-related issues, such as corrupted packet data, but this is all covered by XNA. The drawback of this network library is that it is focused more on Xbox connectivity over desktop applications as players are required to have a Windows Live account to join a network implemented by this library [23]. This restricts the user-base as individuals who do not own an Xbox or a Windows Live PC Game are unlikely to have such an account.

XNA GUI Libraries:

Squid [24]: is the most popular GUI library in XNA, providing a robust and complete solution for GUIs. Advanced functionality is provided, including drag-and-drop UI (user interface) elements, hierarchical opacity, snapping/anchoring/docking windows, z-order layering/clipping/scissoring, and much more that

is uncommon for GUI libraries. Along with its own element skin builder, this provides a very customisable and rich GUI for the developer.

Simple GUI [25]: is the opposite of Squid; it is a library that has gained popularity for enabling minimalistic and simple GUI implementation. It provides the basic functionality required to create a simple GUI in terms of UI elements and controls, such as textboxes and buttons. It does not include any complex elements (such as data grids) or even message dialogs. Using SimpleGUI, a developer can create a minimalistic UI quickly due to the lack of complexity involved in any of its packaged elements.

2.2.2 Pygame Game Engine

Pygame [26] was originally created by Pete Shinners on 28th October, 2000, but became a community project from 2004, with a stable release much later on 6th August, 2009. It packages a number of Python modules intended for the development of video games. Pygame extends an existing library, SDL (Simple DirectMedia Layer), which is written in the C programming language. SDL is a free and open-source library used by developers to access operating-system-specific functions. Pygame acts as a lightweight wrapper, building on SDL, for real-time game development whilst excluding the low-level procedures of C [27]. Pygame is free, with no costs required to either use or commercialise games written with it. It is released under GPL (General Public License), allowing users to create open-source, freeware, shareware and commercial games [28].

Pygame's biggest advantage is that it is truly portable. It supports all Linux distributions, all Windows distributions and BDS distributions [29]. Pygame is community-driven, with plenty of tutorials and example games available with an active user-base. There is also a flourishing library of user-created modules that developers can include in their own projects [30]. Debugging and profiling is slightly more challenging due to the lack of support in hot reloading/swapping with Python [31]. This is where code can be manipulated whilst a Python program is in run-time, which is incredibly useful for game development and provides easier debugging. Excluding this, along with a reasonable amount of documentation, Pygame is a popular choice for 2D game development.

Pygame Network Library:

Twisted [32]: reigns as the go-to networking library for Pygame developers, licensed under the open-source MIT (Massachusetts Institute of Technology) license. Twisted provides all the necessary functionality required to implement various networking systems. Its main purpose is to provide general networking solutions; it is not specifically tailored for games. The library supports a number of network protocols such as SMTP, POP3 and DNS. Alongside this, it is a very flexible library, with the source code and open license (MIT) developers can extend the library and add any feature they need. The most common weakness in the security of network software is buffer overflows, however due to being written in Python, Twisted is immune to this [33].

Pygame GUI Libraries:

pgu [34]: is a simple GUI module and all-purpose library. It packages simple UI elements, making it more suitable for games that do not require complex GUIs. For this reason, any GUIs created with pgu can be easily implemented into existing Python games. A significant advantage to its minimalistic approach is that there are no input handling issues, little overhead and low response time. For this reason, pgu is

great for arcade-type or real-time based games. Although simple, this library is known to produce attractive GUIs and provide rare functionality, for instance, a menu system. However, there is currently no scrolling widget, such as a scrolling textbox, and some features are not yet implemented in a stable release, such as disabling/hiding widgets.

OcempGUI [35]: opposes pgu, providing a relatively high-level complexity to its GUI features whilst retaining all the core functionality expected from a GUI library. As such, it is much more suitable for complex GUI implementation. It is also appropriate for network games as OcempGUI contains module support for the Twisted network library. The library implements layers of high abstraction; customisable GUI elements can be created and manipulated easily. It contains elements that pgu excludes, such as scrolling widgets, multi-line edits and image maps. Unfortunately, with the addition of the aforementioned complex functionality, the response time is lower than other libraries. For this reason, OcempGUI suffers performance issues and is ultimately considered unsuitable for arcade-type or real-time based games.

2.2.3 Slick2D Game Engine

Slick2D [36] packages a number of tools and utilities created specifically for 2D Java game development. It is built on an existing library, LWJGL (Lightweight Java Game Library), for its graphical rending. Through LWJGL, developers gain access to important cross-platform libraries. These include OpenGL (Open Graphics Library), OpenCL (Open Computing Language), OpenAL (Open Audio Library) and an interface to controllers such as gamepads, steering wheels and joysticks [37]. With this abstraction, LWJGL is a simple API that has become the foundation of many popular Java game libraries, including Slick2D. Slick2D extends this by including support for images, animations, particles, sound, music, sprite sheet, fonts and more. As such, the developer is provided with the essential components required to write a game in Java [38].

There are many advantages to using Slick2D. Written in Java, all the code runs in the JVM (Java Virtual Machine) implemented on most operating systems, including Windows, Mac, Linux, BSD and more. This means that a game written in Java can be easily deployed to most platforms, including Android, with very little effort from the developer. There are many deployment methods for Java programs, using the JNLP (Java Network Language Protocol) they can be automatically downloaded and executed. This also allows programs to be embedded into web pages as applets [39]. With games constantly being updated, these methods can ensure that users retrieve the latest version of the game with ease.

The advantage over Pygame, detailed in **Section 2.2.2**, is that Slick2D is created specifically for 2D games. This provides a simple, clean and consistent API to 2D game development, excluding the complexity of 3D development. Opposed to Python, Java supports hot code swapping/reloading, this allows users to debug and manipulate code during application run-time.

Slick2D has a very large active community, with plenty of support, documentation, tutorials and example code available. Overall, Slick2D is a highly supported game library and is very suitable for 2D game development.

Slick2D Network Library:

Kryonet [40]**:** is currently the most popular library for networking in Java game development. It supports both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) communication through a

consistent and straightforward API. Kryonet utilises Java's NIO (new Input/Output API) for network communication, however, Kryonet is specifically for the client-server model. Messages are automatically transmitted over the network as object graphs through the use of Kryonet's serialisation library. A lot of useful additional functionality is included such as threading, LAN (local area network) discovery, logging and RMI (Remote Method Invocation).

Slick2D GUI Libraries:

Swing [41]**:** is the main Java GUI widget toolkit and part of Oracle's JFC (Java Foundation Classes), an API that provides GUIs for Java applications. Because of this, it is an intensively developed and polished library. Swing provides an extensive set of components to develop detailed GUIs. Each widget can be assigned colours, backgrounds, opacities, borders and other visual properties. Although it provides this high level of customisation, it can be difficult to produce GUIs that are usually required in games. Swing is developed as a general GUI development solution, focusing on emulating the appearance and input handling of the platforms the application may be run on, rather than being used in game development.

TWL (Themable Widget Library) [42]**:** is another GUI library for Java but utilises the OpenGL API. The standard set of widgets expected from a GUI library is included, such as labels, edit fields, frames and more. Many advanced features are also supported enabling developers a wide-ranging set of tools to create complex designs. This includes a very powerful HTML renderer that supports XHTML/CSS, including floating elements, images, unordered lists, text alignment and much more. As expected, this allows an impressive level of flexibility that is rarely seen in any other GUI library.

2.2.4 Summary

From the survey on game engines and their respective network and GUI libraries the following table can be used to demonstrate their performance against the project requirements. This table along with the research conducted above shows the necessary information in order to choose the appropriate game engine, network library and GUI library.

Name	Programming Language	Cross-Platform	Open-Source	2D Oriented	No Uncommon Software Required	Web-Browser Support	Compatible Network Library	Compatible GUI Library
Microsoft XNA	C#	×	×	✓	×	×	✓	✓
Pygame	Python	✓	✓	✓	×	×	✓	✓
Slick2D	Java	✓	✓	✓	✓	✓	✓	✓

Table 2.3: Comparison of available game engines against project requirements.

2.3 Networking

This section covers the aspects of networking required to support multiplayer capability for this project. The network protocols available for data transmission, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) will be discussed and then finally the appropriate network models, client-server and peer-to-peer.

2.3.1 Network Protocols

This section discusses the network protocols TCP and UDP. These are original core protocols of the Internet protocol suite and enable the foundation of network programming: the sending and receiving of data.

TCP is a heavyweight, connection-oriented protocol; once a connection is established data can be sent and received bidirectionally. It handles the ordering of packets and also ensures that packets received are identical to those sent, so when a packet is lost or corrupted, the protocol requests retransmission of effected packets. The protocol uses sequence numbers, timeouts and acknowledgements for in-order data delivery, disposing duplicate packets and retransmission of packets along with checksums to assure correctness. The aforementioned functionality guarantees reliability as data ultimately remains correct and in-order. Due to the expensive overhead for this error-checking, TCP is disadvantaged by its speed and as such is not used in cases where time is critical. TCP may also buffer data until there is enough pending before proceeding. This may create problems if the packets need to be sent as soon as possible but are small in size. However, the benefits of TCP have been incredibly important and useful for particular situations where a single action within a game, that may only occur once, cannot be lost. For example creating and joining multiplayer game sessions or a player performing non-movement actions such as swinging at an enemy.

UDP, contrary to TCP, is a simpler, lightweight protocol with no concept of connection. UDP does not handle any errors or provide reliable transmission, so packets may be duplicated, arrive out of order or not arrive at all. There is also no functionality for retransmission, acknowledgements or timeouts. This means that when a packet is sent, there is no way of knowing whether or not it reaches its destination. As UDP excludes the overheads involved in error-handling and reliability, it is much faster than TCP. The suitability for UDP in games depends on the type of message being sent, for instance, actions such as movement and positioning can utilise UDP effectively. These actions need to be sent as quickly as possible but are constantly updated. For this reason, if the game receives an incorrect update or no update at all, this would be later synchronised and rectified from a following update that would be transmitted correctly, allowing the game to continue regardless.

A game can use TCP or UDP; it depends on the type of game. TCP would be more appropriate for event-driven games, where actions sent between network components are scarce and cannot be lost whereas UDP would be more suitable for arcade-type or real-time based games, where constant updates rectify any potential problems in transmission.

2.3.2 Network Models

This section discusses the client-server and peer-to-peer network models that are available to structure the network implementation.

Peer-to-peer: is where each computer involved acts as either a consumer or supplier of resources for every other computer in the network. In video games, the approach to implementing a peer-to-peer network is to abstract the actions within the game into a sequence of instructions. If the state of the game changes for a player, the other players can replicate this change by processing the sequence of instructions in the same order [43]. For example, if a player moves and attacks an enemy, the other clients can simply simulate the same order of instructions from a shared state to update their game. However, with the action genre becoming popular in the industry, a game called 'DOOM' released in 1993, demonstrated the limitations of the peer-to-peer model [43]. Players were able to run the game reasonably well over the LAN, but experienced terrible performance with client connections over the internet [44].

Client-Server: This limitation was overcome by John Carmack, changing the model to **client-server** instead of peer-to-peer when he released the game 'Quake' in 1996 [43]. This model involves two roles, a client and a server. The server acts as the central computer component which distributes the resources within the network, with the client computer communicating with this server for said resources. For games, the server would act as the unconditional correct game state that is replicated for the clients. Similar to the peer-to-peer model, the game is abstracted to a sequence of instructions. However rather than sending the sequence to every other player, it is simply sent to the server, where the server would update its game state and relay the sequence to every other client.

Comparing the two models, the peer-to-peer bandwidth requirement grows exponentially with each player added due to the replication of data being sent to every client. However client-server structure will only require a linear growth for the server with every client added, whilst each client remains at a consistent level of bandwidth. Having a centralised server creates further advantages. The ability to add features such as tokens, filtering and connection logs allows a much greater control over the network. Also, the majority of NAT (Network Address Translation) issues are eliminated. An advantage to the peer-to-peer model is that it is not dependent on a server - it is completely decentralised. The client-server approach introduces a single point of failure - the server. If a player disconnects during a game session in a peer-to-peer based network, the other players can continue, whereas if the server disconnects in a client-server network, no players can continue.

2.4 Summary

After conducting the literature survey the first project objective was met. A list of 25 questions was compiled from various reliable sources and the mini-games were chosen from the studies discussed. Three separate game engines and related software libraries for GUI and networking were analysed. Their strengths and weaknesses were summarised into a table and compared against the requirements for this project. Using this table, it became apparent that Slick2D performed best against the criteria involved. The advantages and disadvantages of the TCP and UDP protocols were discussed. It was found that TCP would be more appropriate for event-driven games and UDP for real-time games. The peer-to-peer and client-server models were compared, finding that client-server had advantages in bandwidth and centralised functionality but suffers from introducing a single point of failure.

3 DESIGN

This chapter covers the main design aspects of the project. The development and network approach used for this project will be detailed. The game overview will be described, explaining the desired gameplay that players experience. With this, the main components of the game are identified, allowing detailed design discussion.

A number of gameplay design principles were considered in the design phase of the project [45] [46]:

- **Meaningful play:** *demonstrate a purposeful relation between a player's action and the game's outcome.*
- **Gameplay balance:** *the reward for each player option should be based on the level of risk taken, preventing dominant strategies.*
- **Provide positive and negative feedback:** *the game must indicate whether a player's action has had a negative or positive effect on achieving their goal.*
- **Flow in Games:** *the state of the game must transition in a natural and consistent manner.*
- **Easy to Learn – Difficult to Master:** *the mechanics of the game should be easy to learn, however consistently achieving top results must be difficult.*
- **Rewarding the Player**: *the game must reward a player's actions accordingly, allowing them to achieve their goal.*

These principles will be referenced throughout this chapter, demonstrating how the design assimilates these factors.

3.1 Software Development Process

This project adopted an iterative approach to development. This enabled the management of the complex systems in the game and separating them into 3 main cycles whilst providing flexibility to any changes. **Figure 3.1** below demonstrates the typical focus on individual procedures throughout a project's iterative development cycle.

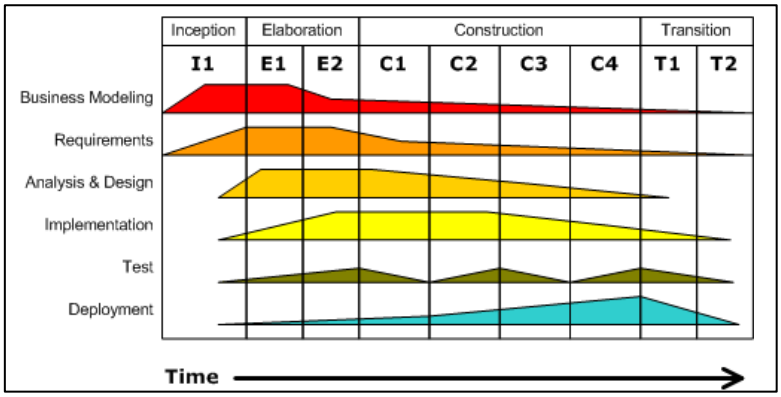


Figure 3.1: Iterative development illustration. **[47]**

The main phase of requirements and design takes place before any development, however using an iterative approach, this continues throughout the project. The major focuses in testing occur after the

end of each cycle but, similar to requirements and design, it takes place during the cycles themselves as well.

For this project, 3 cycles, titled phases, were planned:

- **Initial Phase:** The foundation of the necessary game mechanics are developed, including the essential network functionality. On completion of this phase, the game can test the connection establishment between players and display a main game screen for simple interactive gameplay, such as rolling dice and navigating the board.
- **Main Phase:** The main components of gameplay are created; this involves the questions and mini-games. On completion of this phase, users are able to play through the overall desired gameplay of answering questions, playing mini-games and looping back to the main game screen created in the initial phase. Any necessary changes recognised from the initial phase will also be implemented.
- **Final Phase:** The game's end functionality will be added, allowing the game to terminate. This involves the results screen, feedback and statistics for the user. Any changes necessary for the entire game from the initial phase will be applied to complete the final build of the game.

Using these cycles, any unforeseen challenges can be identified and resolved by revising the phases, providing the important flexibility required in game development. A detailed Gantt chart can be found in **Appendix B** for a detailed plan taken by this project.

3.2 Network Approach

The **client-server** model was the approach selected for implementing the network for the game. As discussed in **Section 2.3**, this would introduce a centralised server that would be very beneficial for certain aspects in this project. This means that the server can record statistics and game logs, providing useful information for improving a player's experience. Another advantage is that it would allow users to play the game immediately. Routers do not need to open or forward particular network sockets and firewalls can remain unchanged, thus avoiding any complex configurations. The network protocol will be **TCP**; this is due to the event-driven nature of the game. There is no real-time movement or object updates required, simply the sending and receiving of a response from the player. This means that the advantage in speed provided by UDP is not necessary. As such, TCP will guarantee that messages will be received in-order and at an acceptable speed.

With the networking infrastructure chosen, we can now detail the game's design.

3.3 Overall Game Design

This section introduces a design of the overall game. Many of the game components and details will be abstracted to provide an overview of the game before going into detail in subsequent chapters.

3.3.1 Gameplay Overview Use Case

An outline of the typical gameplay a player should experience will be explained through a summarised use case scenario for the player in a game session. This will allow us to identify the key components of the game for further requirements and design analysis. The gameplay will be written out as a sequence of events.

1. First, any player can create or join an instance of the game.
2. Once the connection is established players will take turns to choose a character that represents them as their avatar and the game will start.
3. Players take turns in rolling dice, allowing them to navigate a 9 by 9 square board made up of tiles, with each tile representing an activity.
4. There are four different tiles: mini-games and three difficulty levels of questions (easy, medium and hard). The content of these mini-games and questions are discussed in the literature survey**.** There are three different outcomes after both players have taken their turns:

   a. If either or both players have landed on a mini-game tile, both players will then play a mini-game.

   b. If both players land on the same difficulty question tile, both players will then be given the same question.

   c. If both players land on different difficulty question tiles, players will then be given different questions corresponding to the level of difficulty represented by their tile.

5. When a mini-game is played: The players will play one of the four games (Prisoner's Dilemma etc.), the time and process varies depending on the mini-game.
6. When a question is given: Players will be given the same amount of time to answer a multiple choice question of the appropriate difficulty as detailed in step **4c**.
7. After either step **5** or **6**, the results are then displayed and the points are distributed according to the success of either player.
8. The steps from **3-7** are repeated until 15 minutes have elapsed.
9. When the time is up, players are taken to a final screen, where they are shown their rank, scores, achievements, feedback and statistics. The player with the highest score is crowned the winner.
10. The game has now finished, player information is submitted to the server and players can now exit the game.

The following Activity Diagram (Figure 3.2) is a simplified flow of overall gameplay.

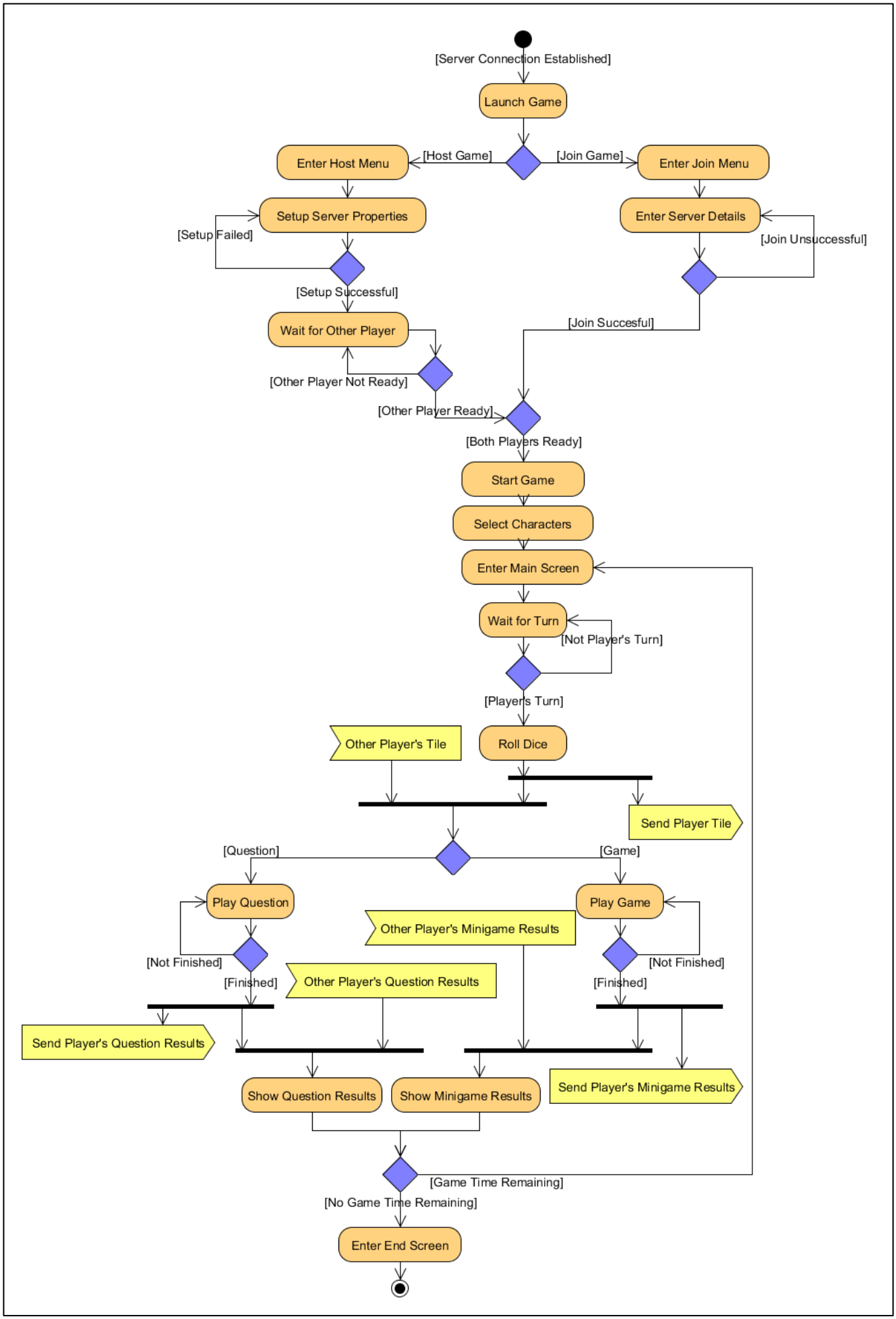


Figure 3.2: Activity diagram of simplified gameplay overview.

3.3.2 Overall Requirements

This section includes the overall requirements that are consistent throughout the game. These requirements have been elicited from the gameplay overview in **Section 3.3.1**, the proposed solution in **Section 1.3** and a preliminary audience survey carried out available in **Appendix A**. Functional requirements are tasks or processes that must be performed by the game. The non-functional requirements are standards or qualities that the game must adhere to.

The requirements have been prioritised using the following scale [48]:

- **Essential**: the game is unacceptable if these requirements are not satisfied.
- **Conditional**: enhances the game, but it is still acceptable if not satisfied.
- **Optional**: a functional or quality enhancement; nice to have if resources permit.

Functional Requirements:

Reference #	Requirement Description	Priority
General		
GFR1	A secure server will host the game server and other system files.	**Essential**
GFR2	The server will support simultaneous instances of two-player sessions.	**Essential**
GFR3	The server will support non-password and password protected sessions.	**Conditional**
GFR4	There will be a tutorial explaining the details of the game.	**Conditional**
GFR5	There will be a ranking displaying the top 10 scores imported from a server file that stores all previous player scores.	**Conditional**
GFR6	Players can navigate through a main menu comprising of “Host Game”, “Join Game”, “Tutorial” and “Scoreboard”.	**Essential**
GFR7	The game will store an avatar selected by players from a list of 20 characters to represent them throughout the game session.	**Conditional**
GFR8	The game client will store and start a countdown timer from 15 minutes once a session has started.	**Essential**
GFR9	The game server will synchronise each player’s game client timer throughout the session.	**Essential**
GFR10	The game will simulate the Prisoner’s Dilemma for a two-player environment	**Essential**
GFR11	The game will simulate the Sealed Bidding Auction with two players.	**Essential**
GFR12	The game will simulate the Trust Game using lock-stepped networking between two players.	**Essential**
GFR13	The game will simulate the Ultimatum Game between two players.	**Essential**
GFR14	The game will simulate quiz based gameplay for players to participate in answering questions.	**Essential**
GFR15	Players will be able to communicate using instant messaging.	**Essential**
GFR16	A scoreboard showing player names and their scores will be displayed throughout the game session.	**Essential**
GFR17	The questions will be stored externally to the game code in HTML documents.	**Essential**

Table 3.1: Project functional requirements.

A further detailed requirements analysis can be found in Appendix C for the functional requirements GFR11-GFR15 (Question and Mini-games).

Non-Functional Requirements:

Reference #	Requirement Description	Priority
General		
NR1	The game client will be runnable on the following platforms: Windows, Linux and MacOS X.	**Essential**
NR2	The game client will run at a resolution of 800x600.	**Essential**
NR3	The game client will support full-screen compatibility.	**Optional**
NR4	The game client will run on web-browsers as a Java applet.	**Optional**
NR4	The game Java applet will be runnable on the following web-browsers: Internet Explorer, Mozilla Firefox, Google Chrome and Apple Safari.	**Optional**
NR5	The question documents will be validated against and comply with HTML industry standards.	**Conditional**
NR6	The feedback documents will be validated against and comply with HTML industry standards.	**Conditional**
NR7	The user interface themes will be validated against and comply with XML industry standards.	**Conditional**
NR8	The fonts used in the game will be of 10pt or larger.	**Essential**
NR9	The notifications in the game will be at the top of the game in flashing text to stand out.	**Essential**
NR10	The winning results will be displayed using a green background.	**Conditional**
NR11	The losing results will be displayed using a red background.	**Conditional**
NR12	Buttons that require attention will be flashing to stand out.	**Essential**
NR13	Button animations will be coloured indicating hover, select or active states.	**Essential**
NR14	The player will be notified to contact the developer if any internal errors consistently occur.	**Essential**
NR15	The game client timers will be synchronised with no more than 500ms delay.	**Essential**
NR16	The game client will run with less than 200mb of RAM used.	**Essential**
NR17	The game will be runnable without any additional software installed other than Java.	**Essential**

Table 3.2: Project non-functional requirements.

3.4 Identifying Main Game Components

In this section the main game components will be identified and summarised.

- **Host & Join Game Component**: allows players to host and join game sessions.
- **Main Game Screen:** displays the overview of the game session: where players roll dice, navigate the board and invoke either a question or game. Instant messaging only takes place on this screen.
- **Question Component:** this is where both players participate in a multiple choice question. They are given the question description and a list of choices to choose from. The summary of the results are displayed here.
- **4 Mini-games**: each mini-game represents one of the games: Prisoner's Dilemma, Sealed Bidding Auction, Trust Game and Ultimatum Game, as described in **Section 2.1.3.**
- **End Screen:** acts as the final screen of the game, displaying the overall results including scores, ranks, achievements, feedback and statistics.

The diagram below shows the flow between the main game components, this demonstrates how they are connected from game start to game end.

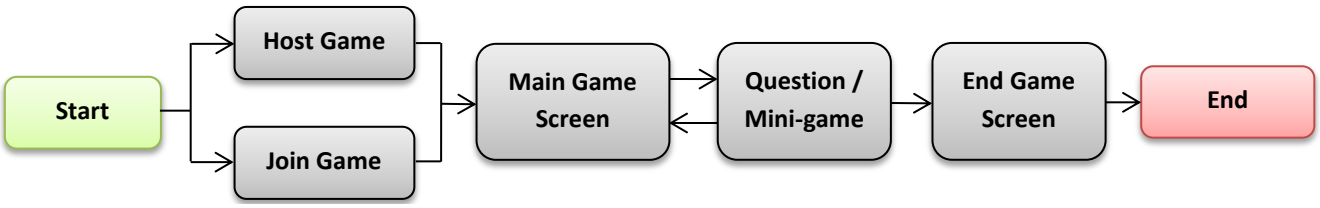


Figure 3.3: Main components flow diagram.

A player will either host or join a game. They will then cycle between the main game screen and answering questions or playing mini-games until the 15 minute time limit has elapsed. This supports the '*meaningful play*' principle; players are given a larger context of the game in the main game screen, whilst competing for points in the smaller components of the game (questions and mini-games). After this they will be taken to the end game screen where they are shown their results and the game ends. After identifying the main components the subsequent chapters will discuss them individually in detail.

3.5 Host & Join Game

As mentioned in the previous section, players must establish a connection using the host and join components. Any player can host a game but only a player with the correct details can join said game. Players are expected to have existing communications in order to discuss the details of the server in order for the second player to join the hosted game.

Host Game: This allows a player to initialise a game session for another player to connect. The player is required to enter a player name but is not required to set a password for the server. If there is no password set for the server, players can join without a password and the server is deemed "Public". If the player chooses to enter a password, the server is "Private" and players attempting to join are required to provide this password. An identifier needs to be generated to represent the game sessions. This will be a unique session ID (an incremented counter), which is then sent to the player.

Join Game: With the game hosted and session ID generated, the first player can message the second player the details of the server, allowing them to join. The player is required to enter a player name, session ID and if the server is private, a password. If the player provides the correct details they will be notified of a successful connection. When the hosting player is ready, the game will start.

The sequence diagram below demonstrates the network design for the connection establishment process provided by the host and join game. With the connection established between the two players and the game started by Player 1, the game component next initialised is the main game screen.

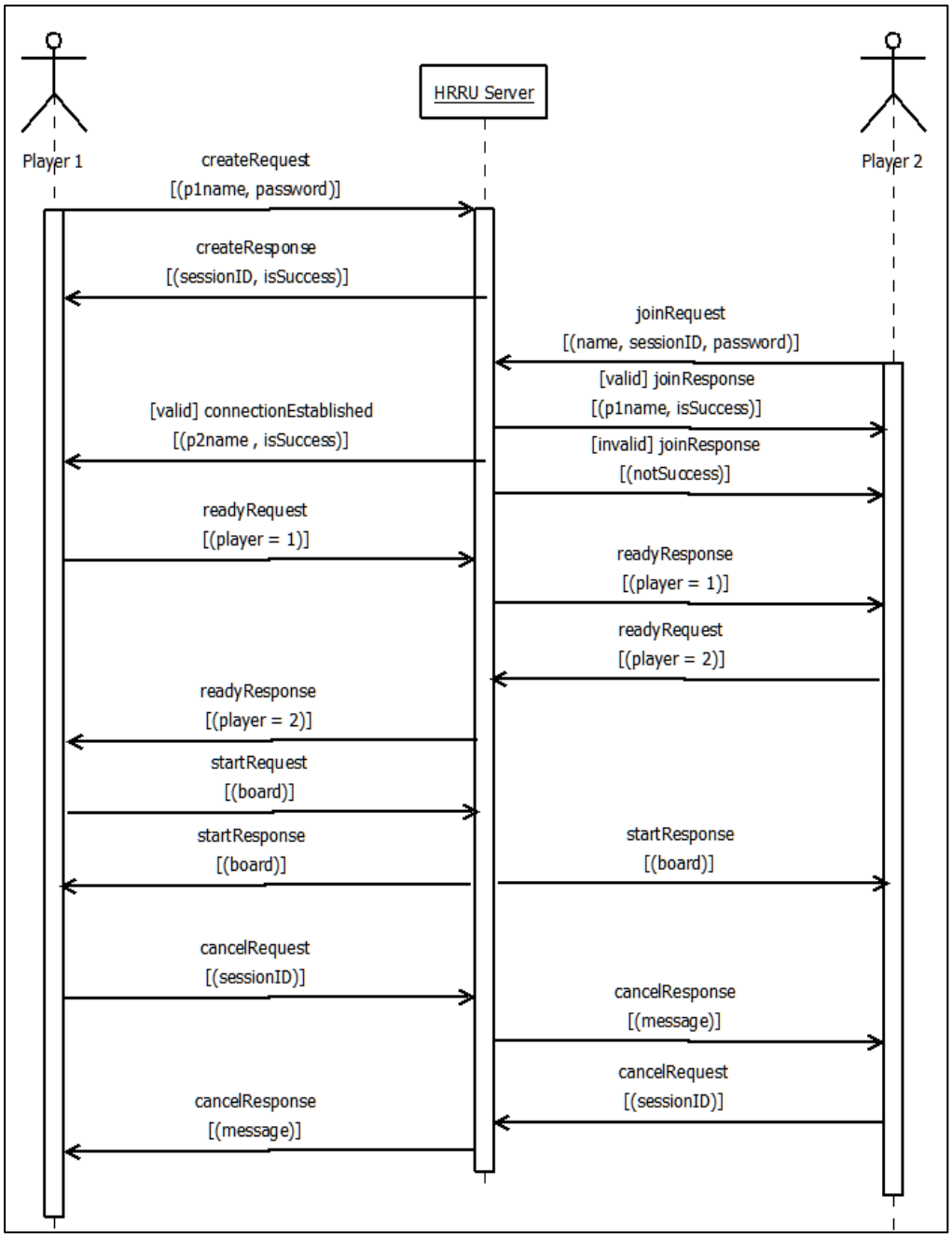


Figure 3.4: Sequence diagram for connection establishment network process.

3.6 Main Game Screen Design

This is the central component to the game, acting as the medium to the question and mini-game components. Players roll dice, navigate the board and invoke either a question or mini-game depending on the tile they land on. As such, this is a complex game component, combining 5 smaller modules. These include:

- **Scoreboard:** listing the player names, avatar images and scores.
- **Header:** a text banner notifying the player of the current game state, for example player turns. This includes the game timer.
- **Roll Dice:** the pseudorandom dice mechanism involving an animation.
- **Chat box:** the instant messaging system allowing players to communicate, this includes a scrolling list box for the messages and a textbox for entering messages.
- **9 by 9 Tile Board:** involves the board itself, a 32-tile square board including 4 different tile images representing easy, medium, hard questions, and mini-games. This board indicates whether players are given a question or mini-game.

The scoreboard supports the principle of *'providing positive and negative feedback'.* Players can view their scores throughout their game in comparison to eachother, allowing them to discern whether their actions are helping them win over the other player. The diagram below displays a simplified GUI design for the main game screen and the layout of the internal modules.

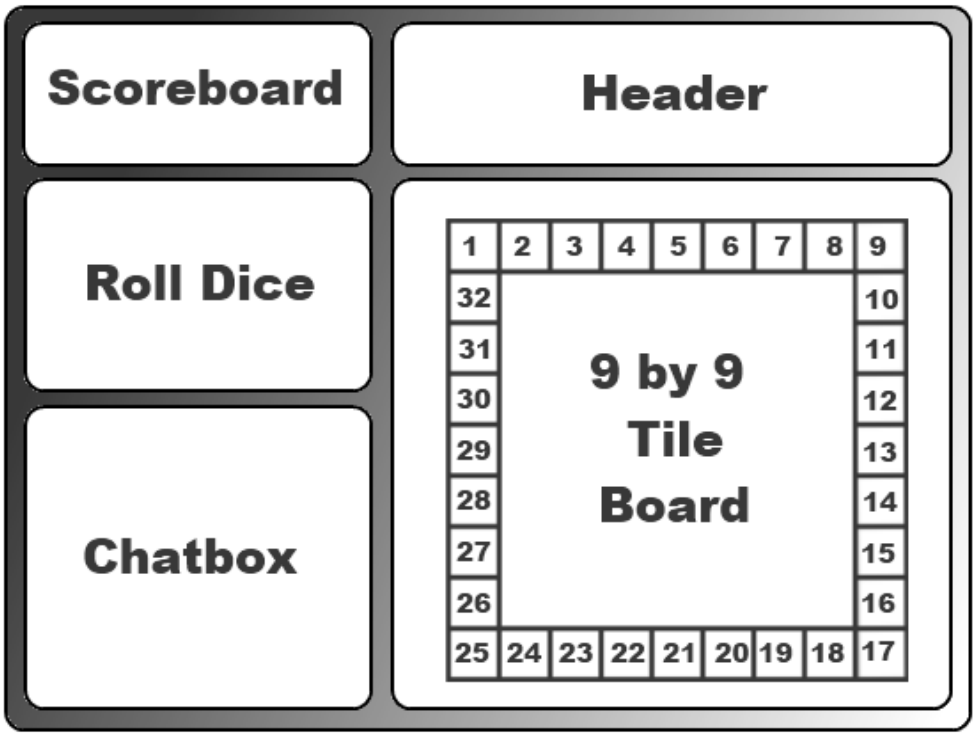


Figure 3.5: Main game screen GUI design.

This approach was chosen as it would provide a unique and enjoyable experience to the questions and mini-games; this was considered the most important aspect for an application measuring rationality from potential users, found in **Appendix A**. Using this board game approach, players are uncertain of what will come next, thus countering the monotonous list of ordered questions in a questionnaire. This also enables replay-ability; every instance of the game will be different due to the pseudorandom board, dice and question/mini-game generation. This approach supports the *"flow in games"* principle; players are introduced to a consistent and enjoyable cycle by navigating the board before playing a question or minigame. The subsequent sections in this chapter will discuss processes involved in the main game screen.

3.6.1 IM Chat System

The IM Chat System facilitates player to player communication; both players can send and receive messages in the main game screen. Players have already established the game session through existing communications – excluding this functionality in the game itself would be inconvenient to the players. Without the IM chat system, players would have to switch between the games and, for example, an instant messaging service such as Google Chat.

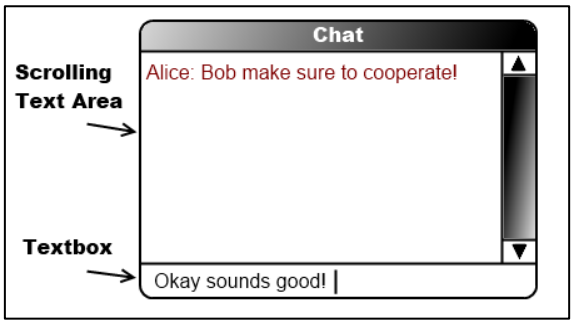


Figure 3.6: Chat box UI design.

As the game is revolved around a competitive environment, the addition of this functionality would provide further advantages. After answering questions or playing mini-games, players may want to discuss their experiences, particularly in mini-games, where teamwork would lead to a greater success. For example, in Prisoner's Dilemma, players may attempt to communicate and agree to both "cooperate", leading them both to succeed.

A simple illustration of the chat box UI design can be seen in **Figure 3.6**. It involves a textbox for players to type and submit their message, and the scrolling list box for the messages from both players to be displayed. Both players' messages will be displayed using a different colour to avoid confusion, making it easier to read.

3.6.2 Navigating the board

Players will take turns to roll the dice and navigate the board, starting with Player 1. Both players will start at the first tile at the top left of the board numbered "1". Player 1 will be notified that it is their turn via the header, and they will be able to roll the dice. This will generate a pseudorandom number (between 1 and 6) and their avatar will be moved around the board clockwise according to this number. Player 2 will then be notified that it is their turn and repeat the process, resulting in both players landing on a new tile on the board. The tile represents either an easy, medium, hard question or a mini-game. The component then invokes either a question or a mini-game; this is the tile establishment phase.

3.6.3 Tile Establishment

As explained in the gameplay overview**,** if either player lands on a mini-game tile, they will then go to the mini-game component. However if they both land on a question tile (easy, medium or hard), they will both go to the question component. The pseudocode below describes the method in selecting which mini-game or question is given:

```
If (either player tile is a mini-game tile)
        enter mini-game that has appeared the least
else if (both player tiles are the same difficulty tile)
        enter same difficulty question
else enter individual difficulty question
```

3.6.4 Synchronising Player Turns

After a player has finished their turn, the other player's game client needs to be updated and synchronised. The sequence diagram in **Figure 3.7** below demonstrates the network design for synchronising the player turns and which question or mini-game is to be played.

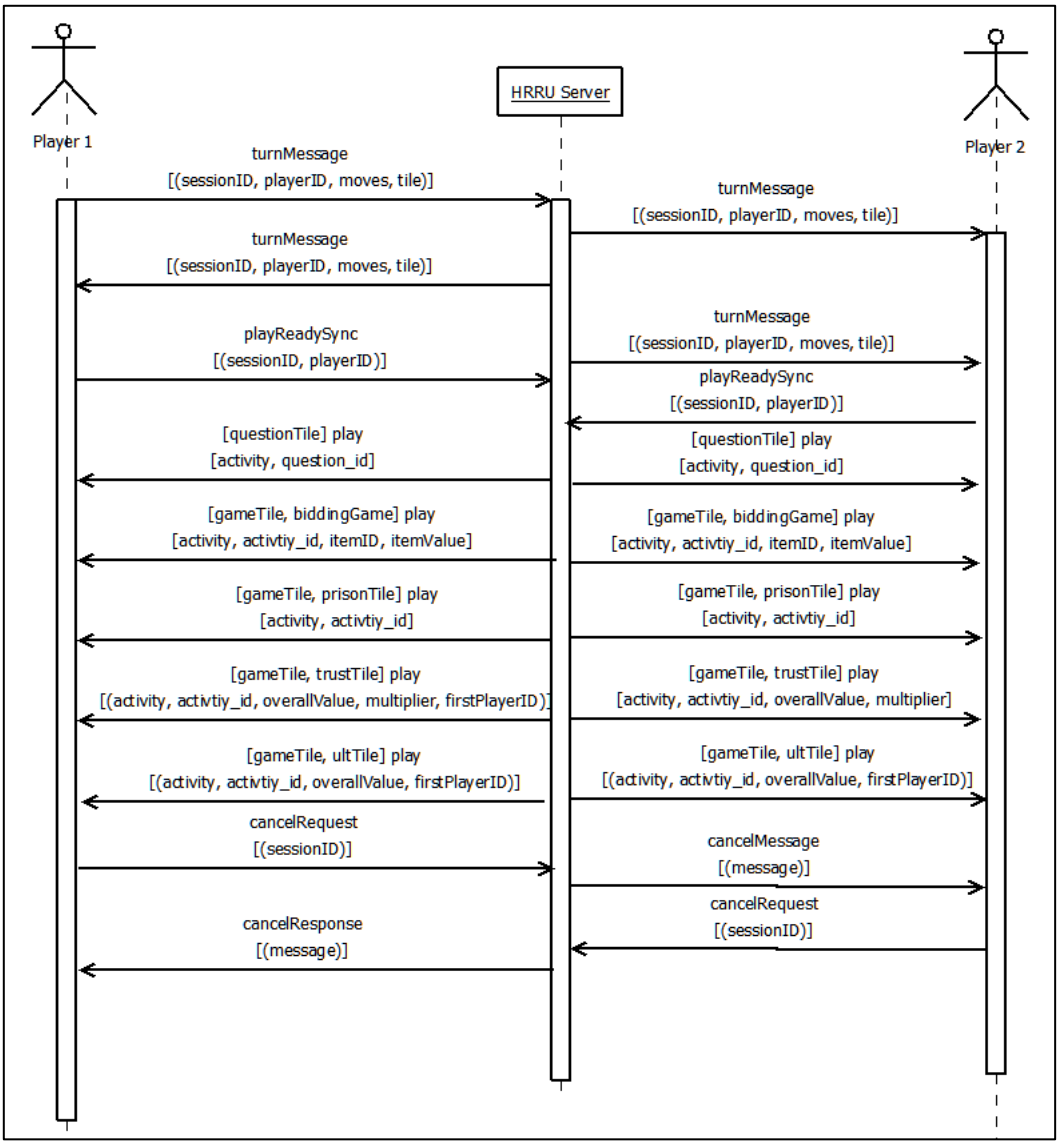


Figure 3.7: Sequence diagram for main game screen.

3.7 Question Component

This section will discuss the question component where both players participate in a multiple choice question. First, it will discuss the decision for multiple choices against open-ended questions, it will then go on to the approach in presenting the question and the scoring involved. Finally, the network process required to implement this component is described.

3.7.1 Multiple Choice or Open−Ended Questions

There are many reasons why multiple choice questions were chosen. First, many of the existing rationality related questions collected this game were multiple choice. Changing this to an open-ended question would affect the validity of the question and it's relation to rational behaviour. A drawback to open-ended questions in software applications is the variability in textual data, where there are numerous ways to express an answer to a question. For these reasons, multiple choice questions were deemed the most appropriate given the game's environment. With the type of question chosen, we can now discuss how they will be displayed.

3.7.2 Presenting the Question

An approach similar to a quiz was used to design the question process. The player's question description is displayed along with a list of choices. Players are both under a time limit to answer the question. If their answer is correct, they win points, and if the answer is incorrect, they win nothing.

This approach is ideal due to the time constraints of the game. If there was no time limit enforced, one player may have to wait a considerable amount of time before the other player has answered. An advantage of using this approach is that the time limit can be used in the scoring system. For instance, a time bonus may be added onto a player's score when a correct answer is given. The diagram below displays a simplified GUI design for this question component. A text area can easily display the question description and each choice is presented by a labelled button.

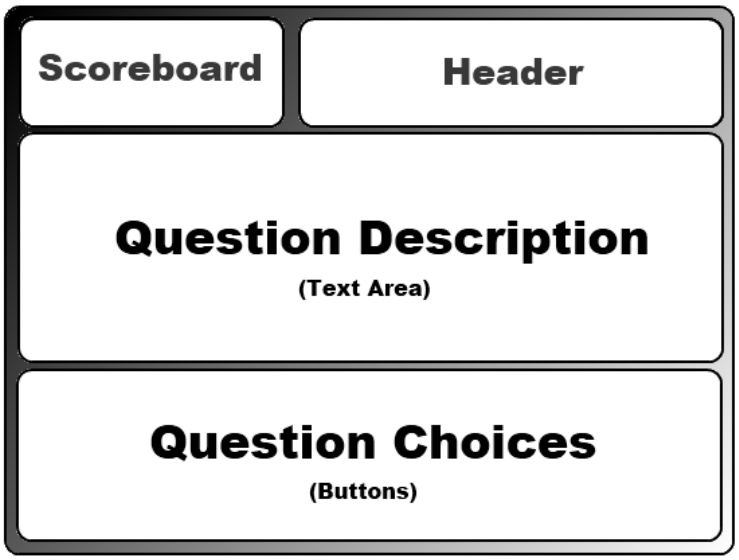


Figure 3.8: Question component GUI design.

3.7.3 Question Scoring

There are various ways to distribute points using this quiz method to questions. There are 2 main variables to take into consideration: the difficulty of the question and the time elapsed. There are a number of different approaches considered for scoring a question.

1. The first approach is by using the difficulty as a multiplier on the amount of points. Each question has a base value of 100 points, this is then multiplied by the difficulty (1 for easy, 2 for medium and 3 for hard), and finally the time bonus (the time remaining after answering the question) is added on top.
2. Another approach is to use the difficulty multiplier on the total amount of points. This means that the time bonus is added to the base value of 100 first, before multiplying by the difficulty.
3. The final approach using the difficulty multiplier is on the time bonus itself. This is where the time bonus is multiplied by the difficulty level, and then added to the base value of 100.
4. An approach using the difficulty level as a constant is also considered, this is where the difficulty level is given a base value of 50, 100 and 150 for easy, medium and hard consecutively. This value is then added to the overall base value of 100, along with the time bonus.

The table below was created using the approaches above against a number of test cases in difficulty and time bonuses. This demonstrates the potential outcomes that may occur in the game.

	Tested Values			Total Points			
Reference	**Base Value**	**Difficulty**	**Time Bonus**	**Approach 1**	**Approach 2**	**Approach 3**	**Approach 4**
1	*100*	*1*	*10*	110	110	110	160
2	*100*	*1*	*60*	160	160	160	210
3	*100*	*1*	*100*	200	200	200	250
4	*100*	*2*	*10*	210	220	120	210
5	*100*	*2*	*60*	260	320	220	260
6	*100*	*2*	*100*	300	400	300	300
7	*100*	*3*	*10*	310	330	130	260
8	*100*	*3*	*60*	360	480	280	310
9	*100*	*3*	*100*	400	600	400	350

Table 3.3: Question scoring approach analysis.

Approach 1 displays a linear increase using the difficulty level and time bonus. With this, the amount of points received for each difficulty level will not overlap. Using Approach 2 the dispersal in points is far too great to provide a balanced distribution, for example, a player can miss out on nearly half the amount of points available by answering 90 seconds later (row 7 and 9). Approach 3 relies on time bonuses; this means that a player who answers a medium question quicker than a player answering a hard question can be rewarded with more points (row 6 and 8), and this is also true for Approach 4. Approach 1 was preferred due to the outcome being directly related to the difficulty level. The game is based on different difficulty levels and the player should feel rewarded when answering a question of a higher difficulty. For these reasons, Approach 1 has been selected as the method in scoring questions.

This scoring system supports the '*rewarding the player*' principle, players gain points not only for answering the question correctly, but the difficulty and time elapsed also contributes to the reward, giving the player a greater advantage for their achievements.

3.7.4 Synchronising Player Answers

This section details the networking involved in the question component using the quiz approach. The sequence diagram below demonstrates the network design for both player's answering their questions and how they are synchronised. Player 1 and Player 2 can answer their questions simultaneously; unlike the main game screen they do not take turns. When a player answers a question, their game client will send a message to the server with their answer, the points they have gained, the elapsed time and their overall new score. The server records these details and then forwards a message to the other player, allowing both players to record the other's results. This enables the results of both players to be displayed on both game clients. After this the server ensures the game clients are still synchronised and the game returns to the main game screen.

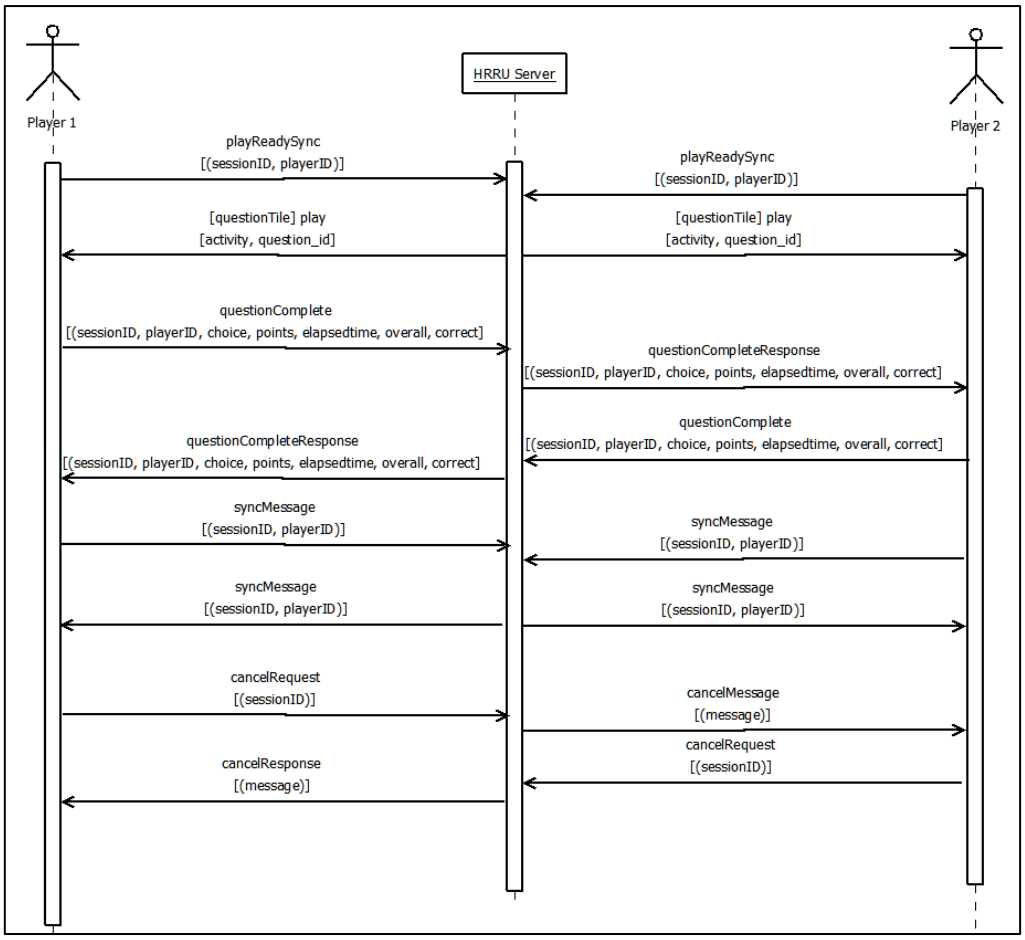


Figure 3.9: Sequence diagram for question component.

3.8 Mini-game Design

This section covers the design involved across all 4 mini-games.

Each mini-game requires a separate user interface and network protocol. Player's will be presented with a description of the mini-game, detailing the steps for the player and the points that are available to win. Player's will be prompted to finalise any choice they have made to rectify any user mistakes. The player will be notified whenever the other player has finalised a choice. Each mini-game also has a specific time limit, when this time has elapsed or both players have finished the required steps the overall results will be displayed. The result screen for every minigame displays the player's results with either a green or red theme (background, text etc.). This supports the '*provide positive and negative feedback'* principle as players can clearly recognise whether their actions have been successful or unsuccessful.

3.8.1 Mini-game Use Cases

The table below includes a simplified GUI design and use case for each mini-game.

GUI Design	Use Case
Prisoner's Dilemma mini-game	
 **Figure 3.10:** Prisoner's Dilemma mini-game GUI design.	Both players are given the same screen, they are required to make a choice to "cooperate" or "betray" within 50 seconds. Players can select their choice by simply clicking the button that represents their decision. Players will not know what the other player has decided until the results are displayed. If both players betray, they receive 0 points. If both players cooperate, they receive 150 points each. If one player betrays and the other cooperates, the betrayer receives 250 points and the co-operator receives 50 points.
Sealed Bidding Auction mini-game	
 **Figure 3.11:** Sealed Bidding Auction mini-game GUI design.	Both players are shown an item to bid for. Players must submit their bid by typing into a text box and submitting the value under a 50 second time limit. The item value represents the number of points available to win, however only the highest bidder wins. If the player has the highest bid, they win the item value in points with their bid deducted.

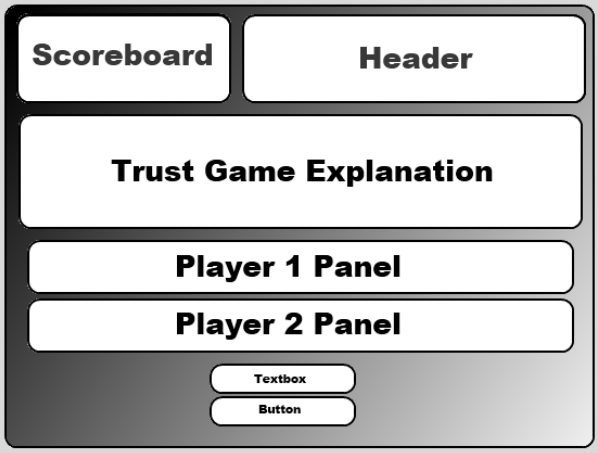
<table>
<tr><th>Trust mini-game</th></tr>
<tr><td> 

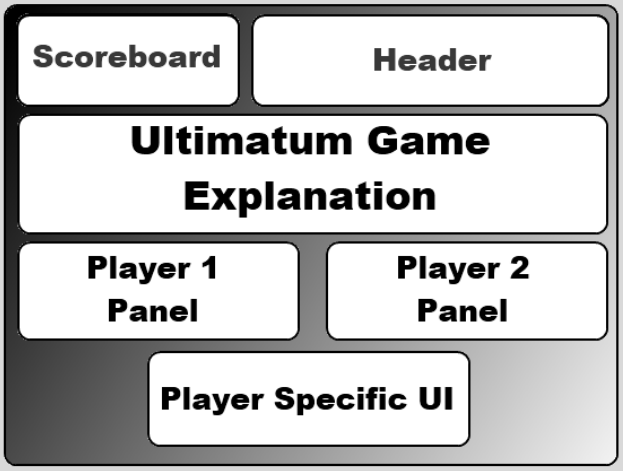
Figure 3.12: Trust mini-game GUI design.</td><td>Players will be allocated the roles: “give” and “receive”. The server supplies a total value and a factor multiplier. The “give” player must choose an amount out of the total value to give to the other player – they will keep the remainder. This amount given will be multiplied by the factor and given to the other player. The “give” player’s panel will then be updated with these values. The “receive” player can now return any of this amount to the other player and keep the remainder. The “receive” player’s panel will then also be updated with these values. Each player uses the textbox to enter the value they wish to submit.</td></tr>
<tr><th>Ultimatum mini-game</th></tr>
<tr><td> 

Figure 3.13: Ultimatum mini-game GUI design.</td><td>Players will be allocated the roles: “propose” and “decide”. The server supplies a total value to be shared amongst the players. The player who is proposing a value will be given a textbox and a button, allowing them to enter and submit the proposed value. The player who is deciding will be given 2 buttons, a button to accept and a button to reject. The player panels display the proposed share for each player.</td></tr>
</table>

Table 3.4: Mini-game GUI designs and use cases.

These minigames allow the player to use various strategies and tactics to succeed. This supports the “*game balance*” principle. Players can only formulate dominant techniques with risk-reward relationships, avoiding any outcomes that may create a gameplay imbalance. This also relates to the *“easy to learn – difficult to master”* principle as players can change their approach to counter another player’s strategy. The minigames are very simple to understand but to prevail over the other player throughout the game will be difficult.

3.8.2 Mini-game Network Design

In this section, the network process designs for the mini-games are briefly discussed. For Prisoner’s Dilemma and the Sealed Bidding Auction mini-game, the studies involve both players answering simultaneously; this is similar to the process involved in synchronising players discussed in Section 3.7.4, with the only difference being the number and type of values sent. The Trust Game and Ultimatum Game are a step locked process, where the other player cannot answer until the other has completed

their turn. This is similar to the process in the tile establishment section discussed in **Section 3.6.4**. Again, the only difference here is the number and type of values being sent. A detailed sequence diagram for each mini-game is available in **Appendix D**.

3.9 End Screen Design

This section discusses the design of the end game screen. First, an introduction will be given, this will include the GUI design and an explanation on the various modules involved. It will then go on to the functional requirements and finally the networking process.

After the allocated time of 15 minutes has elapsed, the main screen menu will enter this component. The end game screen has a number of internal modules, including:

- **Verdict:** An image indicating the rank of the player's score (A to E), 2 achievement images representing certain behaviours the player has demonstrated and a summary of the player's rational behaviour. This follows the *"reward the player"* principle.
- **Question Statistics:** displays the statistics for the questions answered by the player. This includes the average and total points achieved categorised by easy, medium, hard and all questions.
- **Question Feedback:** allows the user to navigate through feedback for each question answered.
- **Mini-game Statistics:** displays the statistics for every mini-game played, categorised by the type of mini-game (e.g. Prisoner's Dilemma). This will include averages, totals and other specific mini-game figures. This will also include the other player's results for comparison.

The diagram below displays a simplified GUI design for the end game screen. The buttons representing the internal modules listed above will simply show a text area containing the relevant module information.

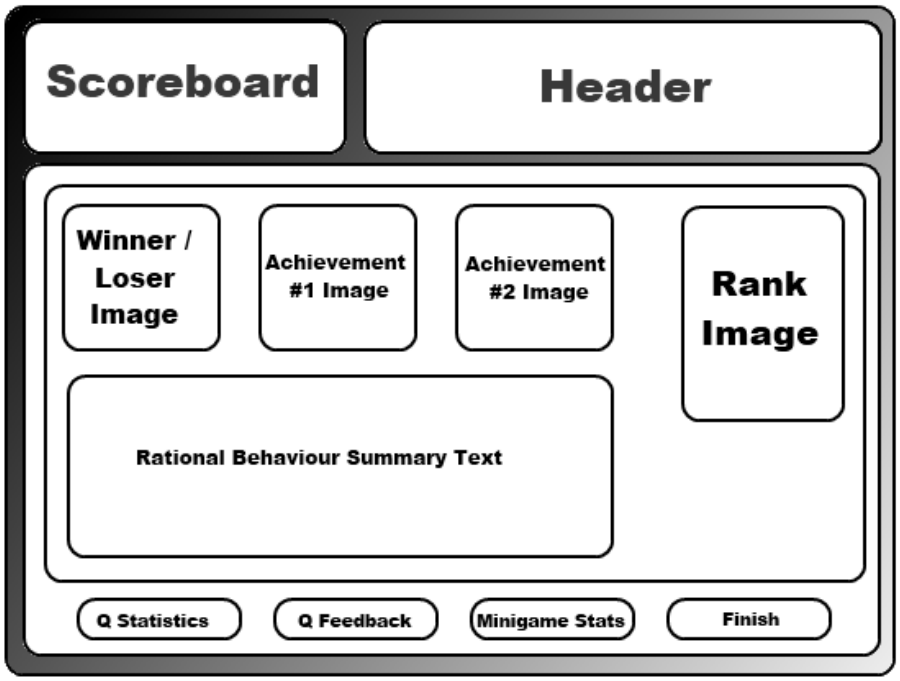


Figure 3.14: End game screen GUI design.

3.9.1 Calculating Player Ranks

The rank will be an important aspect of feedback to the player. Player's will be given a rank from A to E, representing a grading of their rational behaviour. This will be displayed clearly in the end-game screen and uses all the results throughout the game, including both questions and mini-games. There are 2 main approaches considered to calculate this rank.

Fixed Boundaries: this is where the boundaries for the ranks are pre-calculated and fixed throughout every game session. A large sample of results will be recorded from fair test cases and analysed. These test cases will then be used as an indicator of the distribution of points against the number of questions answered correctly and mini-games won.

Relative Boundaries: this approach uses the number of questions/mini-games and calculates a unique set of boundaries for each game session. This will require a fair and balanced algorithm that distributes the ranks according to the points available in each question or mini-game.

The fixed boundary approach would require a great deal of testing with a representative sample of the game's targeted audience. Even after this has been achieved, analysing the results will be very difficult – the ability to distinguish a fair fixed set of boundaries greatly limits the flexibility in gameplay for other players. For example, a player may be considerably slow in answering questions or responding in mini-games, this means that the other player is instantly at a disadvantage as the number of questions and mini-games is less than the average amount used in calculating the fixed boundaries.

Relative boundaries would counter this problem as it is dependent on the game session itself. For this reason, using relative boundaries is the preferred approach. Although, some testing will still be required to ensure that the algorithm is suitable and fair, this would also solve any inconsistences produced by the pseudorandom generation of values that would have been affected by the fixed boundaries approach.

The player ranks support the *"provide positive and negative feedback"* and *"reward the player"* principles. Players are given useful and interesting feedback through accurate grading that is not unfairly affected by the other player's actions.

3.9.2 Game Session Network End

This section details the final networking process leading to the game's end. The sequence diagram below demonstrates the network design for the terminating functionality - saving the final results and scores. Both game clients will automatically send the end scores to the game server. The game server will then save these results and update the scoreboard, sending the new top 10 scoreboard to both game clients. The game clients will then send all the results recorded throughout the game, this involves the mini-game and question scores along with the player's answers to questions. This enables the server to calculate detailed statistics for balance testing.

The game clients will then terminate by sending the feedback questionnaire responses to the game server, which is then recorded to support end-user acceptance testing.

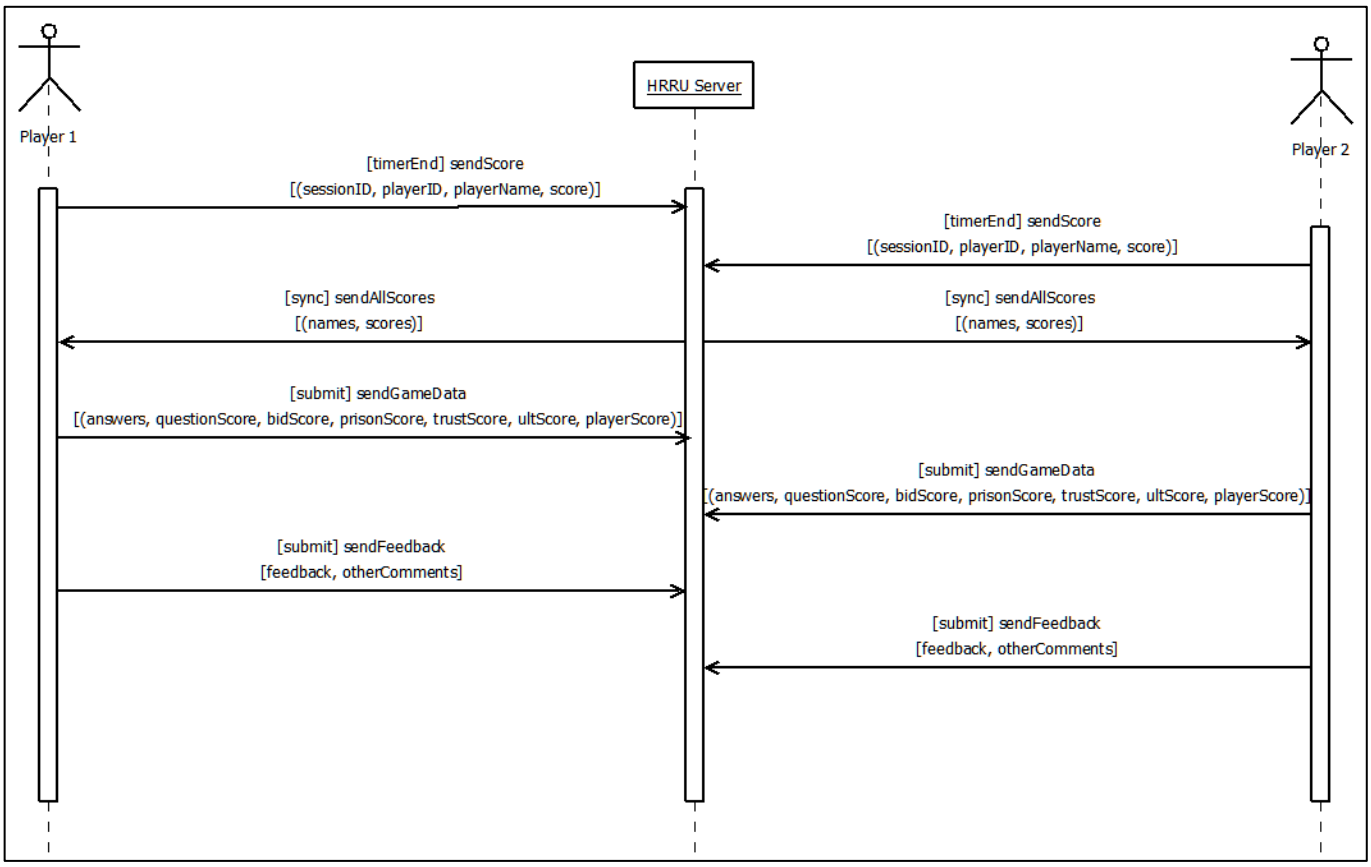


Figure 3.15: Sequence diagram for end game screen.

3.10 Summary

To summarise, this section has detailed the necessary designs to implement the proposed solution.

An iterative approach will be adopted for this project's development and testing. The network model has been selected, declaring client-server as the more appropriate choice. The desired gameplay has been established, allowing us to identify the main components of the game: **Host/Join Game**, **Main Game Screen**, **Question**, **Mini-game** and **End Screen**. Each component was then described in detail, including various functionality, approaches and decisions made. Finally, the game design principles have been effectively met throughout the game thus supporting the implementation of good gameplay.

With the game design covered, the report will now continue to the implementation phases of the game.

4 IMPLEMENTATION

This chapter covers the development approach, the selected technologies and the implementation of the important modules in this project. The modules discussed are either vital to the functionality of the project or particularly difficult to create:

- **Network:** includes the game server, client and session that enable online multiplayer capability.
- **Board Game System:** the board creation and the tile structures.
- **Question System:** includes the structure of the question files, how they are stored and how they are displayed.
- **Feedback System:** involves rank, achievements and statistics calculations.

Problems met during this phase will be discussed, detailing certain objectives and requirements that were not met and why. Finally, a summary will be provided on the implementation of the project.

4.1 Art Assets

The majority of art assets used in this game have been created from first principles. All other media has been retrieved from the Open Game Art Resources [49] website under GPL (General Public License), allowing developers to use the licensed content freely. All content used has been credited and used appropriately, adhering to the GPL legal requirements.

4.2 Selected Technologies

This section describes the choices made for the technologies in the project.

Slick2D was chosen as the 2D Game Library to handle the game-related tasks of this project, including graphics, animations and the game loop. This was mainly due to the engine meeting all the main properties required by the project, displayed clearly in **Table 4,** Section 2.2.4. This includes deployment on all common platforms whilst being open-source and 2D oriented. There is no uncommon software needed to run games created by Slick2D; users only require Java to be installed, which is usually installed as default on many systems. This means that it has the benefits of being written in the Java programming language. This facilitated support for hot code swapping/reloading for efficient debugging and also future deployment opportunities with the possibility of embedding the game into web pages and porting to Android.

Another important factor is that Slick2D was easily integrated with other modules that were desired for the project including the Kryonet networking library and TWL GUI library.

Kryonet, was chosen as it provides a great level of customisation for the NetworkListener class and the structuring of messages. It is particularly adept at client-server based networking as the majority of core functionality required to implement a client-server network is provided. **TWL** was chosen as the GUI library as it facilitates an incredible amount of customisable content. It features functionality that is especially useful for this project, in particular, the HTML renderer. With this, all the questions and their answers can be softcoded into HTML documents, separate from the game code. The possibility of

incorporating separate UI themes is also desirable as user convenience is an important aspect of any game.

To summarise, the game will be created using **Slick2D** for game-related tasks, **TWL** for applying the GUI and **Kryonet** for developing the **client-server** network. With the necessary technologies selected, the implementation of the game will now be discussed.

4.3 Network Capability

The network is comprised of 3 main components, the game session, the game client and the game server. The class diagram in **Figure 4.1** shows the main network classes in each component and their relations. The game server is an executable Java program which implements a network listener for any incoming data on a TCP port. With this, the game engine's network client can establish a TCP connection using the server's address and opened port. When a client connects and hosts a game, the server creates an instance of the *GameSessionObject*. This records the two player's connections with the session ID using a HashMap. This is used to track a centralised state of the game between the two players, such as player board positions and other session details. The game clients store and manage a *GameSessionState* (similar to *GameSessionObject*) to track the individual player's current game state. The game client also implements a network listener, listening for any incoming data from the server. This allows the server to communicate to the game clients, updating the game engine's *GameSessionState* with the updated centralised *GameSessionObject* details. With these classes implemented, the server can synchronise the game client's game session efficiently.

To help support multiple simultaneous instances of the game, the server stores and manages 2 more HashMap structures:

- **connections** HashMap<*Connection c, Integer sessionID>:* This HashMap stores each player connection with a game session. This means that when the server receives a particular packet from a game client, the connection can be used to find the game session the client belongs to.
- **PlayerConnections** HashMap<*Connection c1, Connection c2>:* This HashMap stores both player connections from a game session. With this, the server is able to communicate to the other player in the game session after receiving a packet from a client connection.

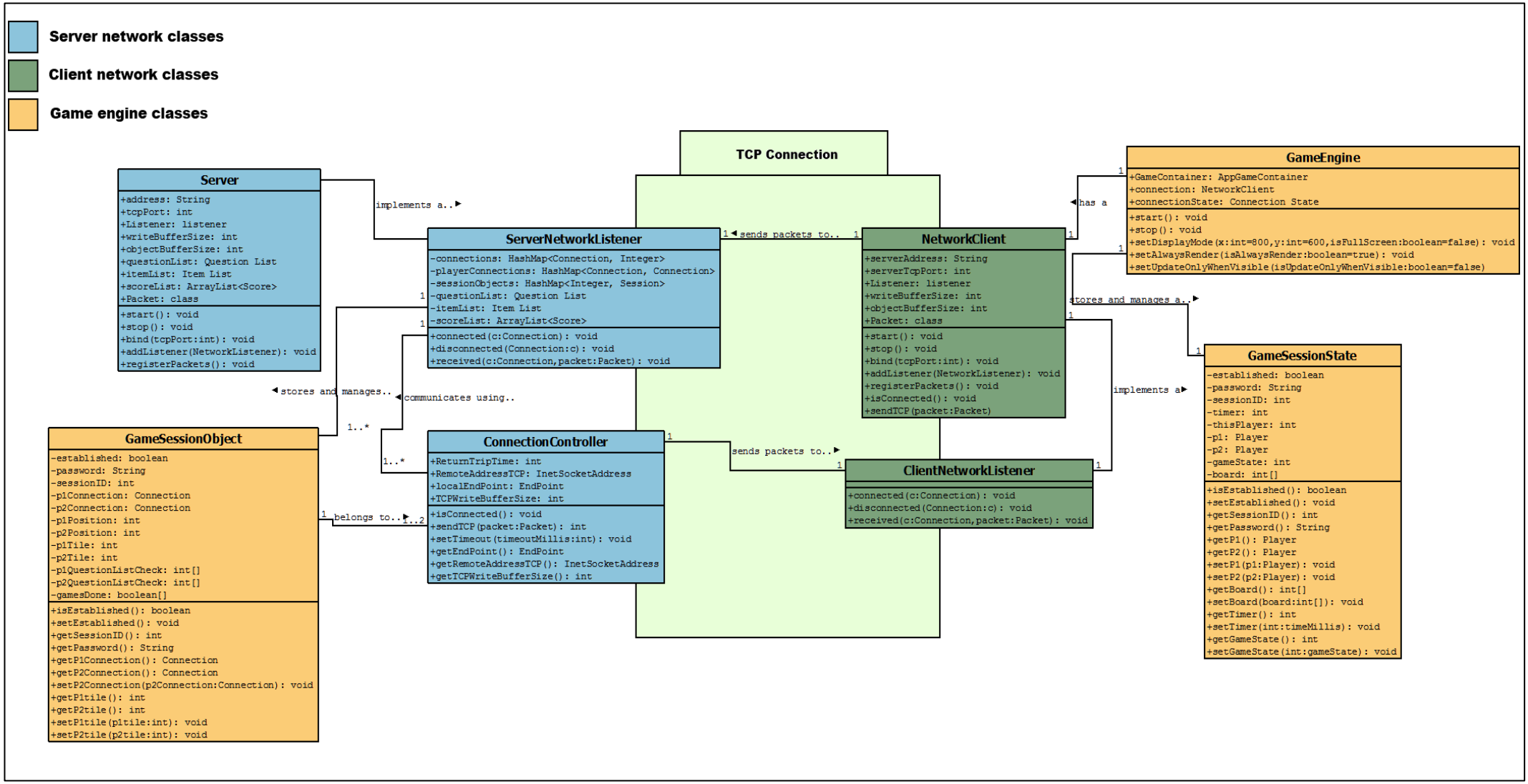


Figure 4.1: Client-server network class diagram.

```
Packet00SyncMessage { int sessionID; int player; }
Packet0CreateRequest { String player1Name; String password; }
Packet1CreateAnswer { boolean accepted; int sessionID; String password; }
Packet2JoinRequest { String player2Name; int sessionID; String password; }
Packet3JoinAnswer { String player1Name; int sessionID; String password; }
Packet4ConnectionEstablished { String player2Name; }
Packet7Ready { int sessionID; int player; }
Packet8Start { int sessionID; int[] board; }
```

Figure 4.2: Host/Join packet class code snippet.

The network uses serialisation to transfer object graphs between the server and its connected clients. **Figure 4.2** displays the packet classes used by the Host/Join components to establish the game session. Both server and client must register these classes using the *registerPackets()* method, enforcing the same packet structures. This will allow the endpoints to communicate using these packets efficiently. For example, the game client can simply use the *sendTCP()* method to send a request to host a game with an instance of Packet0CreateRequest:

***connection**.sendTCP(new Packet0CreateRequest(Bob, 1234));*

This will send a message to the server to host a game using the established TCP connection with the password "1234" and the player's name as "Bob". The server can then easily deserialise this message to recognise the *Packet0CreateRequest* class, access the values and initialise the game session.

This game required a total of 27 packet classes to facilitate the various network processes in the game discussed throughout the design chapter. These packet classes can be found in **Appendix E**.

Synchronisation is achieved using the *Packet00SyncMessage* class. Each game client simply sends this packet when synchronisation is required. The server does not continue until it has received this packet from both players. When this occurs, the server uses the *synchronized* Java function, ensuring that the messages are sent without interference. This enables both players to enter a question, mini-game or the verdict screen at the same time. This is also used to synchronise the game client timers.

With all of the above, the game can provide synchronisation and complex network sequences with using different packet structures, thus meeting the third project objective: ***To implement client-server architecture that allows synchronisation and handles various network sequences successfully.***

4.4 Board Game System

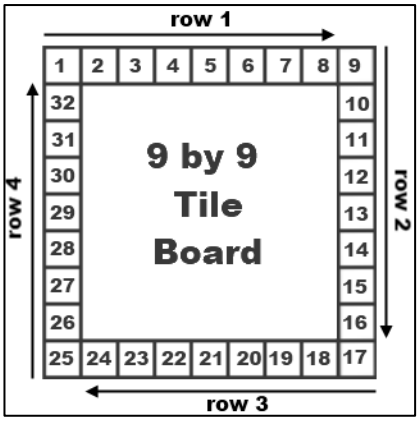


Figure 4.3: Board tile creation order.

The board system was a challenging component to develop. First, the server pseudorandomly generates an array of 32 integers with values from 0 to 3, representing the tile type (easy, medium, hard or mini-game). The server ensures that there are 8 of every tile.

When the game is started, the server sends this array to each game client to ensure both players are using the same board. The game client then uses this 32 integer array to create the board, using the values in the array to specify the tile type. When creating the board, the tiles are stored in the same order

they would be visited. To do this, the board was created row by row, so that the tile positions can be updated in the x-axis or y-axis accordingly, as shown in **Figure 4.3**. The code snippet below in **Figure 4.4** demonstrates how this is implemented for rows 1 and 2.

```
for(int row1 = 0; row1 < scale-1; row1++)  {
    gridSquares[counter] = new GridSquareContainer(tileOrder[counter], shiftx+tempxpos, shifty);
    tempxpos+=GridSquare.width;
    counter++;
}
for(int row2 = 0; row2 < scale-1; row2++) {
    gridSquares[counter] = new GridSquareContainer(tileOrder[counter], shiftx, shifty+tempypos);
    tempypos+=GridSquare.height;
    counter++;
}
```

Figure 4.4: Board creation code snippet.

A simple class diagram of the board game system is shown below in **Figure 4.5;** this includes the structure involved in creating a tile-based board game. A detailed version of this class diagram can be found in **Appendix F.** The board is automatically generated using the 32 integer array, named *tileOrder*, by creating an array of 32 *GridSquareContainers*, named *gridSquares*. The integer variable *scale* is used to represent the size of the board, this means that the board can be increased or decreased in size easily whilst still maintaining the square shape displayed in **Figure 4.3**.

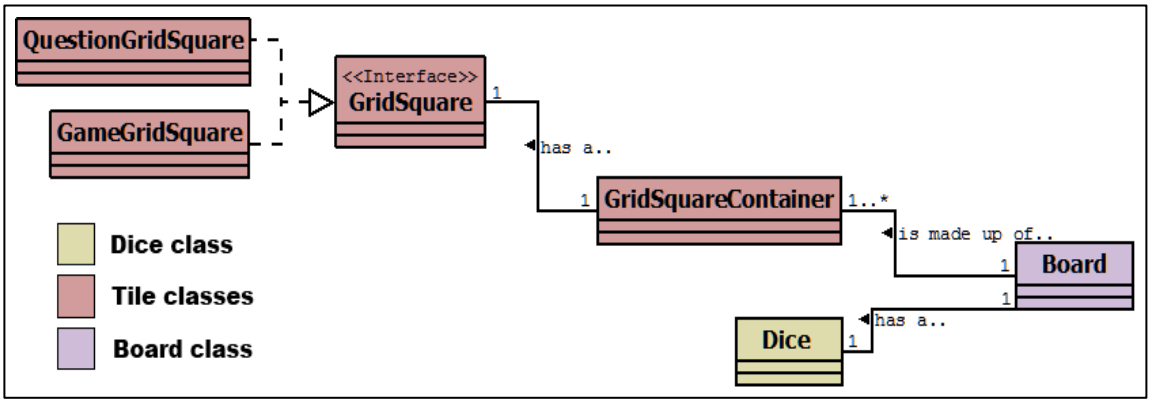


Figure 4.5: Board game system class diagram.

When a *GridSquareContainer* is initialised, it either stores a *QuestionGridSquare* or *GameGridSquare* depending on the tile type given from the *tileOrder* array along with its position on the screen. If the *QuestionGridSquare* is created, it further specifies whether the tile is easy, medium or hard by storing the correct difficulty level and relevant image:

Figure 4.6: Board tile images used.

With each tile stored with its type, image and position, this approach not only enables the game engine to easily draw the board on the screen, but it also allows the player's avatar to be drawn on the board

by accessing the *GridSquareContainer's* position using the player's new index. For example, if the player's new position is tile 4, we can access the *GridSquareContainer* with index 4, to retrieve its coordinates on the screen. The player's position is simply updated by drawing the avatar on each tile from its current position to its new position for a certain amount of time, to simulate the player moving on the board. This in turn also provides whether the player has landed on a question or a mini-game, depending on the tile type of the player's new index.

4.5 Question System

This section discusses the implementation of the question system. This is directly related to the fourth project objective, listed in **Section 1.3.3**. The project objective required a scalable game engine with all questions and feedback to be softcoded. This means that properties such as the question description, the answers and the difficulty level should not be in the game code. This proved to be a difficult objective due to the wide range of questions involved, for example, different question description lengths and formats, varied answer structures, and some questions using images. It was also necessary for this information to be easily manipulated and added. To meet the objective, this question system was created.

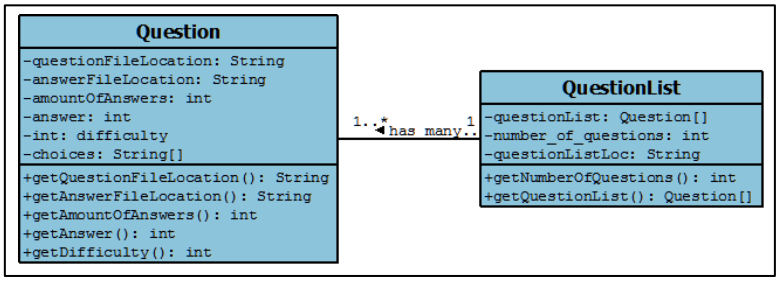


Figure 4.7: Question data structures.

The data structures necessary to store questions can be seen in **Figure 4.7.** These variables and methods provide access to all the properties of a question necessary to implement the system. To avoid hardcoding these properties, the system uses a number of external files written in HTML, CSS and XML. The connection between the data structures and the external files is shown in **Figure 4.8** below.

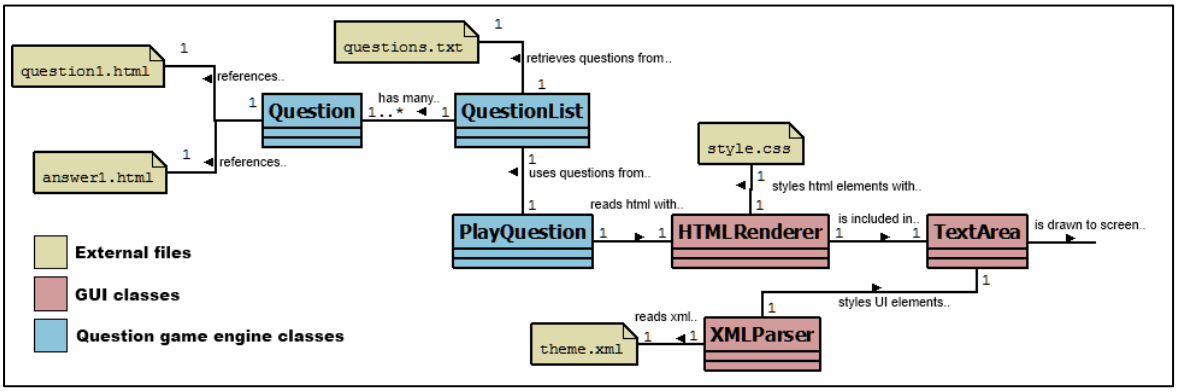


Figure 4.8: Question system structure.

The structure diagram above demonstrates how a question is retrieved from external files and drawn to the player's screen. First, the *QuestionList* data structure retrieves all the properties for each question from the *questions.txt* file; this includes the file location for the question description and feedback. The

QuestionList uses this to create a *Question[]* array where, for every set of question properties retrieved, a *Question* object is created. Every *Question* object can then reference separate HTML files that can distinctively detail the question description and feedback, in this case, *question1.html* and *answer1.html*. The *PlayQuestion* class represents the question component designed in **Section 3.7**, where players participate in answering questions. However, to display these HTML files in Java, the HTMLRenderer class is used to parse the HTML code. This also facilitates CSS code, allowing the HTML to be styled using the *style.css* file. After this is done, the question description can now be displayed using the TextArea widget provided by TWL (Themable Widget Library). This widget also uses an external file, *theme.xml.* This file describes the visual properties for all UI elements and widgets used in the game. A XML parser reads the XML code in *theme.xml* and uses the visual properties defined for the TextArea. This allows the widget to be displayed according to the properties specified correctly, for example, enabling vertical scrolling and fixing the TextArea size.

The table below briefly walks through an example question; this includes only the main steps for brevity:

Step 1: Retrieve properties from *questions.txt* and create *Question* Object.

Text Retrieved	Explanation
7	**Question ID**
4	**Number of Answers**
1	**Difficulty Level**
1	**Correct Answer**
q7.html	**Question Description File**
a7.html	**Feedback Description File**
The horizontal lines are of different length.	**Choice 0**
The horizontal lines are of equal length.	**Choice 1**
The image is not symmetrical.	**Choice 2**
None of the above.	**Choice 3**

Step 2: Retrieve question description HTML and CSS from *q7.html* and *styles.css*.

```
<html>
<head>
    <link type="text/css" rel="stylesheet" href="style.css"/>
</head>
<body>
    <div>
        <div style="margin-left:170px;"><img src="q7.png" style="margin-left:200px; vertical-align:middle"/></div>
        <div style="margin-left:110px;"><p>Which of the following statements is true?</p></div>
    </div>
</body>
</html>
```

Step 3: Retrieve XML visual properties for TextArea widget.

(This displays minimal code detail for brevity.)

```
<theme name="textarea" ref="-defaults">
                <param name="maxWidth"><int>700</int></param>
                <param name="maxHeight"><int>300</int></param>
                <param name="fonts"><map>
                    <param name="default"><fontDef filename="questionatari12.fnt" color="#FFFFFF"/></param>
                    <param name="link"><fontDef filename="Font/font.fnt" underlineOffset="1">
                        <fontParam if="hover" underline="true"/>
                </map></param>
                <param name="images"><map>
                    <param name="ul-bullet"><image>textarea.ul-bullet</image></param>
                    <param name="q7.png"><image>q7.png</image></param>
                </map></param>
                <param name="mouseCursor.link"><cursor>cursor.finger</cursor></param>
    <param name="maxWidth"><int>730</int></param>
    <param name="maxHeight"><int>200</int></param>
</theme>
```

Step 4: The question can now be drawn to the screen from the Question Component class (*PlayQuestion*).


Table 4.1: Step by step question system example walkthrough.

Here we can see that the question is correctly displayed automatically; the description has been formatted, the image included and the answers displayed using buttons. This means that in order to add a question, all that is required is to create a separate question HTML document and update the *questions.txt* file. As shown above, this is very easy; a few lines were added to *questions.txt* and just 3 lines of HTML code written to display the question correctly. This of course means that manipulating question information is equally as simple.

The *style.css* and *theme.xml* files act as a central component for styling all questions and UI elements; this takes advantage of TWL's capabilities. Simply changing a property in the *TextArea* widget XML code will affect all questions, allowing for consistent control over the UI visual presentation. As such, GUI themes can be integrated into the game, allowing players to choose certain pre-sets for the UI (font size, colour and so on).

All of the above demonstrates how powerful the question system can be. Question and feedback information can be added and manipulated easily with no changes being made to the game code at all, thus fulfilling the second project objective: ***To develop a scalable game engine that will allow questions and their corresponding feedback information to be softcoded, ensuring that this information can be added and manipulated with ease.***

4.6 Feedback System

This section covers the broad feedback system that has been implemented for the game. The system is similar to its initial design in **Section 3.9** however it has been split into further separate components: Overall Verdict, Question Feedback, Question Statistics and Mini-game Statistics (with sub components for each mini-game). The separated components provided more screen space and specific structures for the content. The calculation of rank, achievements and statistics is discussed, demonstrating that the complexity of this system comes from the sheer amount of data involved.

4.6.1 Rank

This section explains how the relative rank boundaries were calculated to determine the player's rank. The overall verdict page that displays the rank uses the following frame:

Figure 4.9: Verdict screen interface frame.

To calculate the rank image, the relative boundaries must first be calculated. These are unique to each game session using the points available in every mini-game and question played. This required a very comprehensive approach.

First, data structures were created to store the individual results of each mini-game and question played. An array of these objects was created: *QuestionScore*[], *BidScore*[], *PrisonerScore*[], *TrustScore*[], *UltimatumScore*[]. These were then used to calculate the totals and averages over each mini-game type and all questions, stored in *QuestionScoreOverall*, *BidScoreOverall*, *PrisonerScoreOverall*, *TrustScoreOverall*, *and UltimatumScoreOverall objects*. The variables and relations of these classes can be found in **Appendix G**.

The overall scores achieved by the player can then be compared against the overall points that the player could have won. This percentage then places within the following ranks, displaying the appropriate image and also producing the first sentence that is written in the *Summary Text*:

Rank Image	Boundary	Summary Sentence
S	90-100%	You achieved highest rank: SUPER. You've demonstrated an exceptionally high level of rational and logical thinking!
A	70-89%	Great Job! You've demonstrated a very high level of rational and logical thinking!
B	60-69%	Good Job! You've demonstrated an above average level of rational and logical thinking!
C	50-59%	Not bad... You've demonstrated an average level of rational and logical thinking!
D	40-49%	You can do better... You've demonstrated a below average level of rational and logical thinking, however this could be at the other player's fault, maybe try against someone else?
E	0-39%	Hmm... the results weren't very promising. According to the results you've demonstrated a low level of rational and logical thinking; however this could be at the other player's fault, maybe try against someone else?

Table 4.2: Rank class information.

4.6.2 Achievements

The winner and loser images are simply shown depending on the player scores at the end of the game. However the 2 achievement images are a bit more complex to implement. First, the achievements were created with a relevant image:

Achievement Image	Description	Achievement Image	Description
	Player achieved the most points.		Player bid high amounts in Sealed Bidding Auction mini-games.
	Player achieved the least points.		Player answered questions correctly.
	Player cooperated in mini-games.		Player answered questions quickly.
	Player did not cooperate in mini-games.		Player rolled high numbers during the main game screen.

Table 4.3: Achievements information.

The percentage of each achievement was calculated using the aggergations mentioned earlier. For example, the High Bidder percentage uses the total player bids against the total item values in mini-games both won and lost, the code snippet for this can be seen in **Figure 4.10**.

```
if((biddingScoreOverall.getBsItemValueTotalL() + biddingScoreOverall.getBsItemValueTotalW()) > 0)
    bidderPercentage =  percentage(((((biddingScoreOverall .getBsPlayerBidTotalL()
        + biddingScoreOverall .getBsPlayerBidTotalL()) / ( biddingScoreOverall .getBsItemValueTotalL()
        + biddingScoreOverall .getBsItemValueTotalW())));
```

Figure 4.10: High Bidder achievement calculation code snippet.

All achievement percentages are then stored using a *ResultPercentage* class, this includes the achievement name, image, percentage and description. This class implements *Comparable <ResultPercentage>* and overrides the *compareTo* function:

```
@Override
    public int compareTo(ResultPercentage resultPercentage) {
        int percentage = ((ResultPercentage)resultPercentage).getPercentage();
        return percentage - this.percentage;
    }
```

Figure 4.11: ResultPercentage's compareTo() function code snippet.

This allows an array of achievement percentages to be created using *ResultPercentages[] and* sorted using the Java function *Arrays.sort().* The game then picks the top 2 percentages and sets this as the player's achievements. This implements relative boundaries for not only the ranks, but for achievements as well, avoiding many of the problems of fixed boundaries discussed in design **Section 3.9.1**.

4.6.3 Statistics

As discussed above, with these aggregations a number of statistics were calculated and displayed for each mini-game type and questions. These present an in-depth analysis to the player's results and also provide a means in balance testing. The total numbers of statistics calculated were: 21 for Questions, 19 for Sealed Bidding Auction, 22 for Prisoner's Dilemma, 14 for Trust Game, and 17 for Ultimatum Game, nearing a total of 100 different values. This was the main reason for separating the final screen into the separate components – displaying these results on the same screen would be challenging. After separating the components, a *DialogLayout* widget from the TWL library was used. This allows text to be split into any amount of columns and rows. With this, the following structures were created, using the Prisoner's Dilemma mini-game statistics as an example:

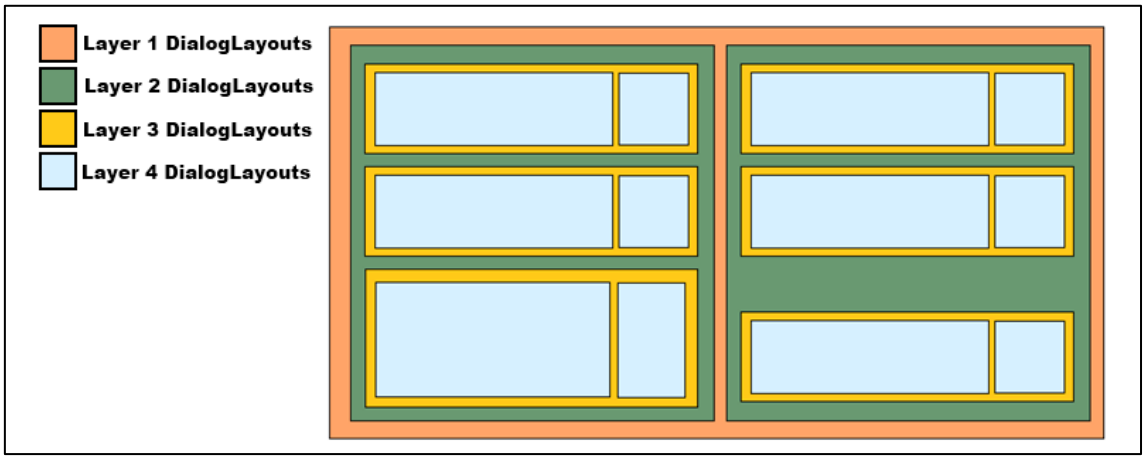


Figure 4.12: DialogLayout for Prisoner's Dilemma statistics.

Here we can see 4 layers of DialogLayouts:

- the first layer is a single-cell *DialogLayout*,
- the second layer is a 2-column *DialogLayout*,
- the third Layer involves 2 separate 3-row *DialogLayouts*,
- finally the fourth layer uses a 4-column *DialogLayout*.

This produces the following result:

Figure 4.13: Screenshot of Prisoner's Dilemma mini-game statistics.

Using this layered approach, the statistics can be structured clearly for every mini-game and question.

4.7 Problems

This section discusses the major problems encountered during the implementation phase. The integration of the TWL GUI library and the problems in deploying the game as a Java applet will be discussed.

4.7.1 Integrating TWL GUI Library

Integrating the TWL library into the Slick2D game engine proved difficult. The Slick2D game engine has its own set of Input Handling and Rendering classes. Rendering the user interface elements created using TWL was not a problem; this was because TWL's render class simply draws on top of Slick2D's rendering. This was not the case with the TWL Input Handling class. Input Handling manages the input made by the player, including mouse and keyboard actions. The problem was that when TWL was integrated into the game, all the inputs would only go to the TWL Input Handling class. Therefore input that should have gone to the game engine was lost.

To solve this issue I contacted the creator of the TWL library, Matthias Mann, through e-mail and explained the problem. He had notified me that the easiest solution would be to leave the TWL Input Handling class on top of Slick2D as it currently is, but to integrate another class in-between them. This was the TWLSlick class; any input that was not consumed by any UI widget from TWL (like a textbox or button) would be forwarded to the Slick2D game engine's input handling methods. The architecture of these classes can be seen below in **Figure 4.14**.

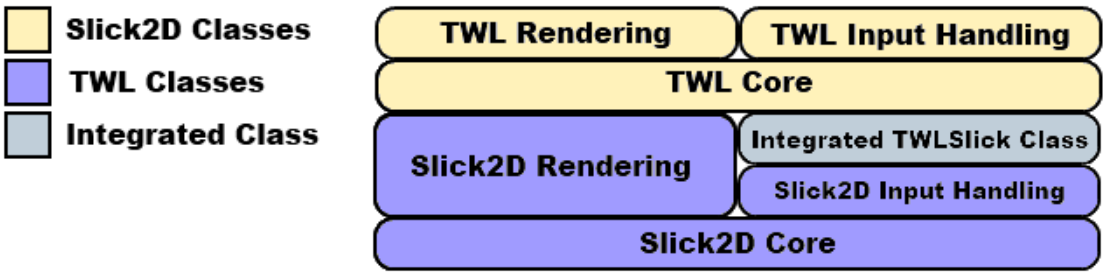


Figure 4.14: Integrating TWL library components into Slick2D.

With Matthias Mann's guidance, TWL was successfully integrated into the Slick2D game engine allowing UI elements created by TWL and game-related functionality to work simultaneously.

4.7.2 Java Applet

This problem also relates to an external library used, in this case, the network library Kryonet. One of the advantages for choosing the Slick2D game engine was that the games created can be deployed as a Java Applet. Unfortunately the difficulty in this was far greater than expected with the integration of the Kryonet network library. The first problem was that the game client needed to be hosted on a webserver. This was achieved using XAMPP, an integrated server package of Apache. Although the game client was now running on a webserver as required, the TCP connection to the game server was frequently timing out. Troubleshooting proved difficult due to the relatively small Kryonet community and unfortunately I was unable to deploy the game as a Java applet whilst retaining the network functionality.

I decided to contact the creator of Kryonet, Nathan Sweet, explaining the problem. After an extensive length of time Nathan was able to rebuild part of his library, solving some of the errors we came across. However we were unable to fix the overall problem and I was not able to afford more time on this, as the priority on the Java applet non-functional requirement was optional. For this reason, the Java Applet

deployment is still a consideration for future development but was ultimately abandoned for this project.

4.8 Summary

To summarise, a number of complex systems were created to handle the game mechanics in a standardised manner:

- The network uses structured packet objects to ensure the game server and clients were able to easily serialise and deserialise data sent and received. The use of these packets provided a simple mechanism in simulating various network processes, achieving the third project objective.
- The Board Game System managed the creation of the board and the tiles used. This enabled for the board to be pseudorandomly generated with ease for each game session whilst synchronising the player's game clients.
- The Question System is used to handle all the questions and feedback involved in the game. It allowed for all the information to be softcoded and excluded from the game code, meeting the fourth project objective.
- The Feedback System managed all the information required to provide the necessary feedback to the player at the end of the game. This included the rank, statistics and achievements.

With the implementation phase covered, the results of the final build version for this project will now be presented.

5 RESULTS

This section will display a walkthrough of the final build game using screenshots. It will illustrate the gameplay and how the players can interact with the system. The screenshots enable the analysis of whether the requirements have been met and demonstrate the broad level of content developed with polish and robust gameplay. The screenshots preceding the Main Game Screen have been taken from player 1's perspective.

5.1 Main Menu

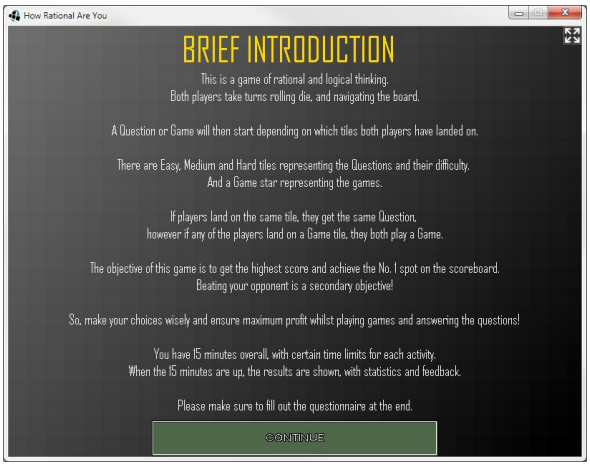
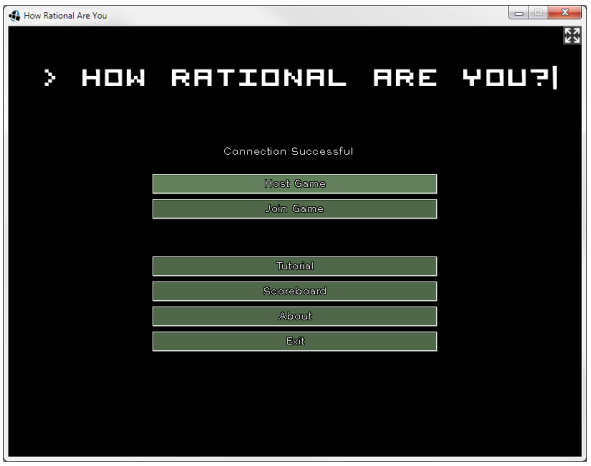
1	Brief Introduction	2	Main Menu
			
3	**Tutorial**	**4**	**Scoreboard**
			

Table 5.1: Main menu walkthrough.

When the game is launched, a brief introduction explaining the game is given before the user reaches the main menu. The main menu provides simple navigation through the use of buttons. The tutorial describes the game in its entirety through screenshots and descriptions. The scoreboard displays the current top 10 scores achieved by players. From here, a user can either host or join a game.

5.2 Host & Join Game

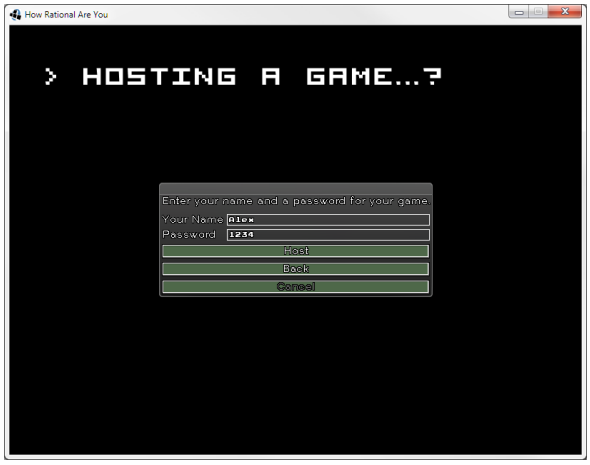
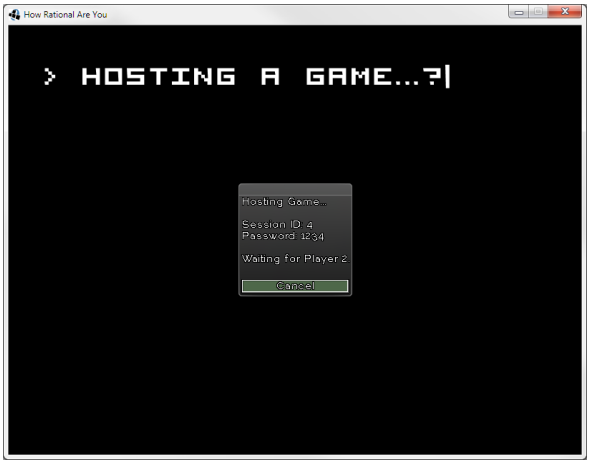
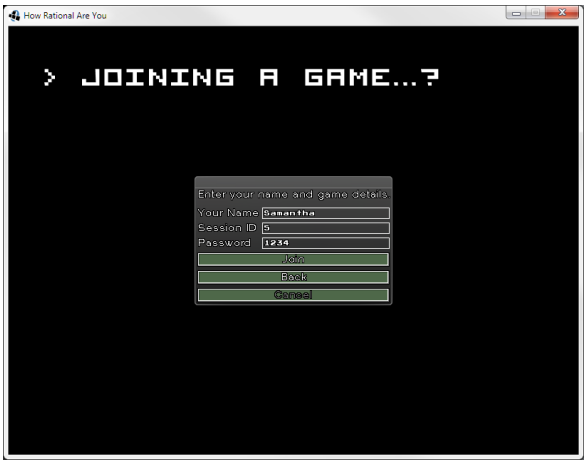
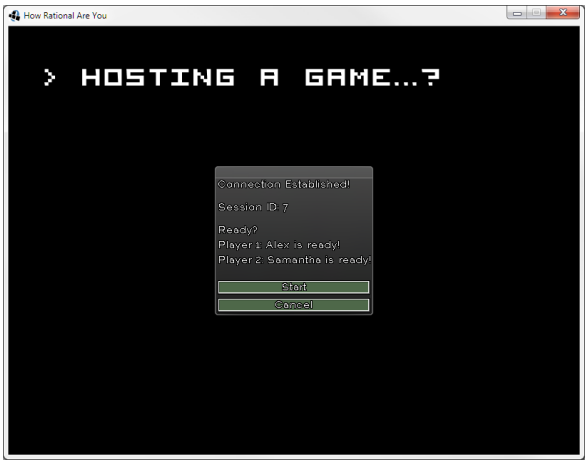
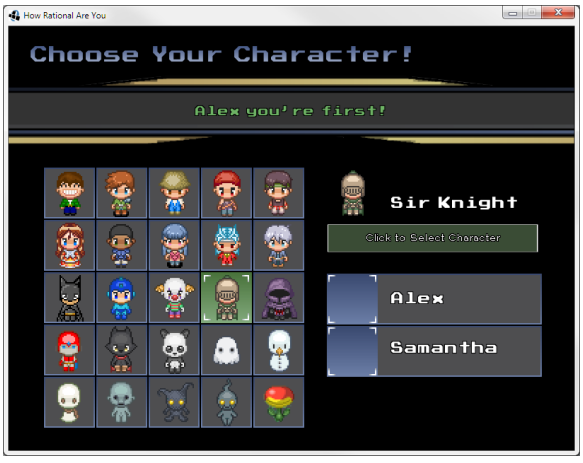
1	Hosting a game	2	Waiting for player 2
			
3	Joining a game	**4**	Ready to start
			
5	Character selection		
		In this section we can see the process involved for players to establish a connection and initiate a game session. One player is able to host a game using their specified details. The second player uses these details from the first player to join the game. Both players then indicate they are ready and the game starts. Following this, players take turns in selecting a character to represent them throughout the game (as seen in step 5). Throughout these steps we can see that buttons are disabled and highlighted appropriately, along with coloured text and banners to notify players of the game state.	

Table 5.2: Host and Join walkthrough.

5.3 Main Game Screen

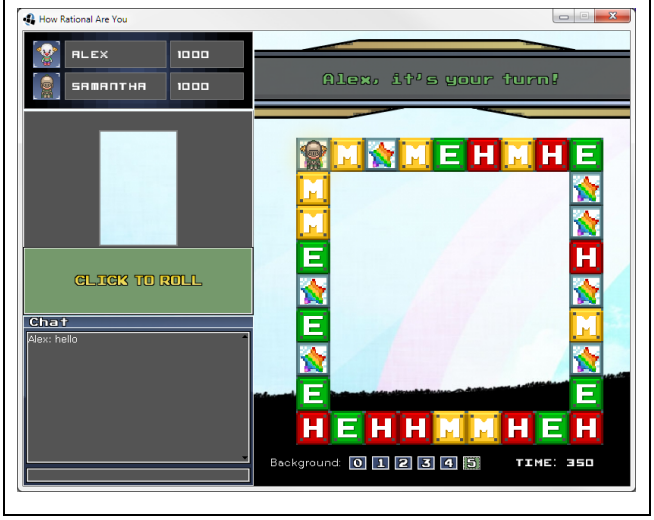
1	Main game screen
	On this screen players take turns rolling the dice by clicking the button labelled “CLICK TO ROLL”, this flashes green to alert the player. After this, their avatar is navigated around the board (seen on the top left tile). Players can use the bottom left chat box for instant messaging. The bottom right presents the overall game timer in seconds. The top left displays the scoreboard containing player names and scores. The top right banner is the header, displaying the game’s current state. A customisable background can be selected using the 6 buttons labelled 0 to 6 at the bottom of the screen.

Table 5.3: Main Game Screen walkthrough.

5.4 Question Component

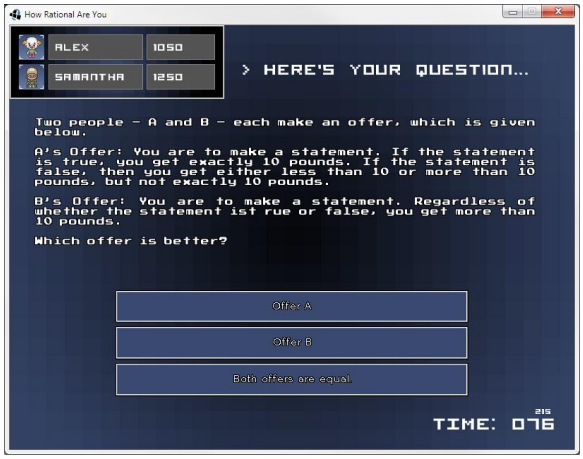
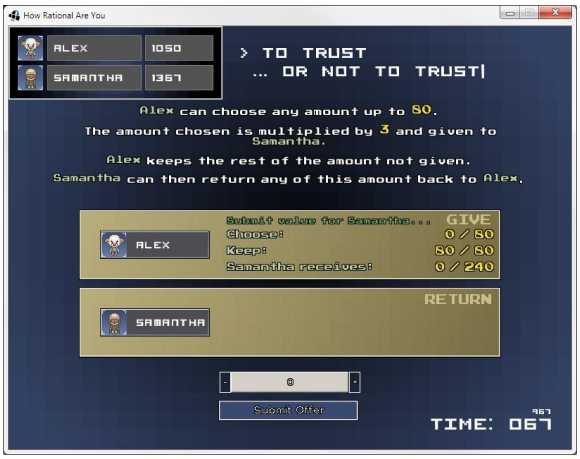
1	Answering a question	2	Question results
			

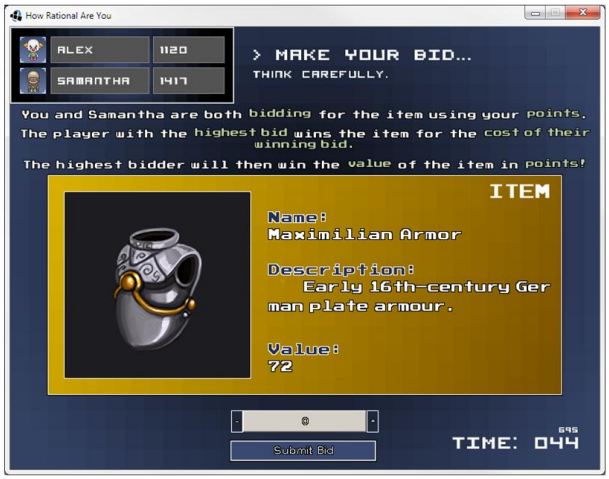
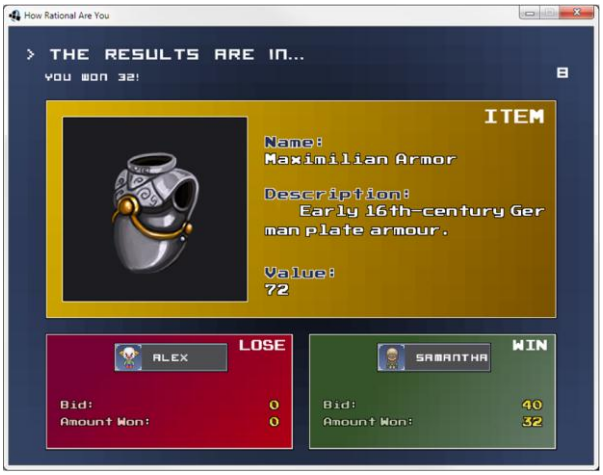
Table 5.4: Question Component walkthrough.

Players can select the button representing their choice to answer the question. Here we can see the scoreboard is still displayed and the timer is given in milliseconds and seconds, simulating a countdown. The question description is displayed clearly in the middle of the screen. After both players have completed the question or the time has elapsed, the results are shown. If players answer the question correctly, a green background is given. If players answer incorrectly, a red background is given. This clearly indicates the outcome of the question. The point distribution is detailed, including the difficulty and time bonus. After this, the game continues, looping back to the main game screen.

5.5 Mini-game Component

1	Playing Prisoner's Dilemma mini-game	2	Prisoner's Dilemma results
			

1	Playing Trust mini-game	2	Trust mini-game results
			

5	Playing Sealed Bidding Auction mini-game	6	Sealed Bidding Auction results
			

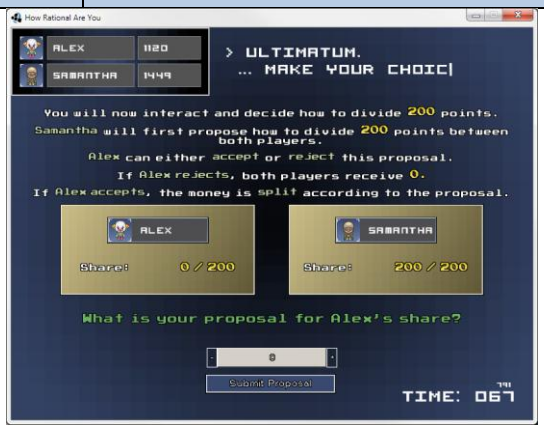
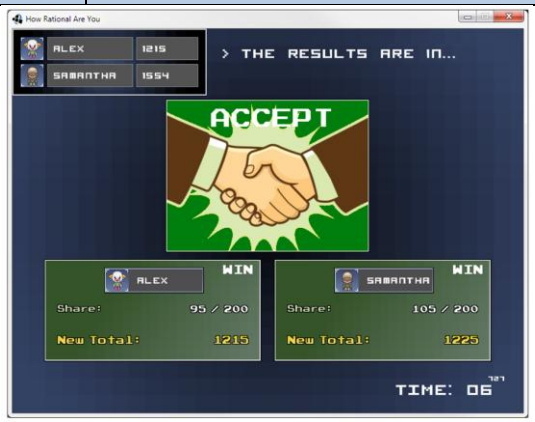
7	Playing Ultimatum mini-game	8	Ultimatum mini-game results
			

Table 5.5: Minigames walkthrough.

This is the mini-game component, including all 4 mini-games: Prisoner's Dilemma, Sealed Bidding Auction, Trust Game and Ultimatum Game. Each mini-game has a separate interactive UI but a consistent overall structure. Similar to the question component, a header is provided at the top right, scoreboard on the top left and a timer on the bottom right. When the player has achieved a successful outcome, a green background is given. If the player achieved an unsuccessful outcome, a red background is given. This clearly indicates whether the player has made a profit or loss in points.

5.6 End Screen Component

The screenshots in this section display the final screen the users can navigate through – an end game menu. The verdict screen shows the achievements, ranks and a summary achieved by the player. Players can navigate through Question Feedback, Mini-game Feedback, Scoreboard or Finish. Question Feedback involves statistics and answers of the questions that have appeared in the game. Mini-game Feedback displays statistics of all mini-games played separated by type. The question feedback contains details of the question, including the correct choice, the player's answer and other details. Players can then view the scoreboard and check whether their score reached the top 10. After this, players terminate the game by clicking "Finish", answering the questionnaire and exiting the game. At this point, the server stores all the results and scores achieved throughout the game session.

1	Verdict	2	Question Statistics
			

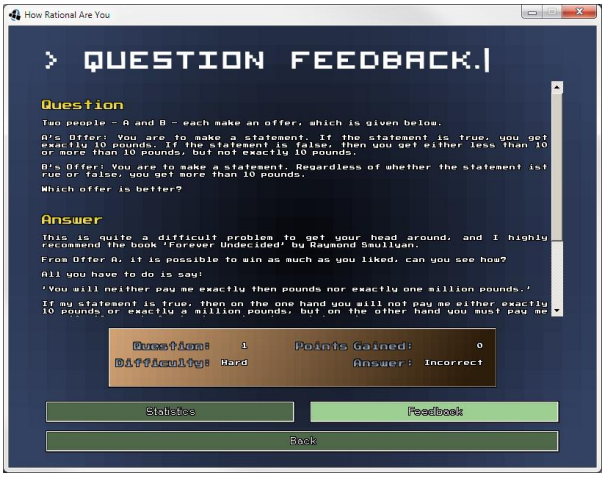
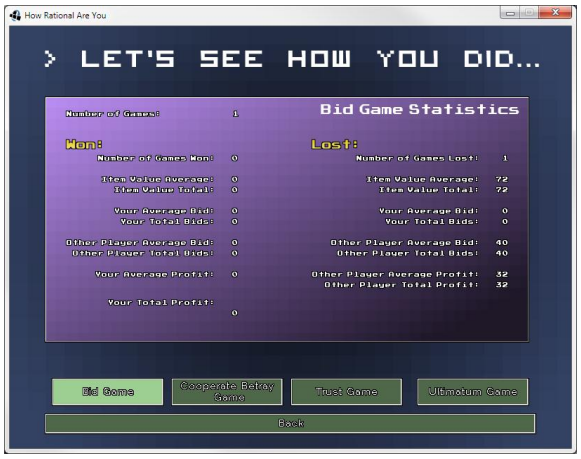
3	Question feedback	4	Sealed Bidding Auction statistics
			
5	**Prisoner's Dilemma mini-game statistics**	**6**	**Trust mini-game statistics**
			
7	**Ultimatum mini-game statistics**	**8**	**Feedback questionnaire**
			

Table 5.6: Verdict walkthrough.

6 TESTING

This chapter will cover the testing carried out for this project. This includes testing procedures, balance testing and end-user testing.

Result sets were automatically processed as the server records the results of every game session completed. This enabled the results to be easily retrieved in a standardised format enforced by the server. These results were used throughout testing and will be discussed in this chapter.

6.1 Bug Reporting

Bug reporting was carried out throughout the testing phases. A public online spreadsheet was created for users to access and edit. Individuals involved in testing were provided a link to this document and asked to write out details on any bugs or other related problems they came across. The format for reporting included an incremental ID, the date, the game version, the game state (e.g. mini-game, question), a brief description of the bug, how to replicate the bug and the user's initials. This produced a centralised and standardised bug-listing. With this, fixing the identified problems within the game became very straight forward. I was able to prioritise the reported bugs and strikethrough those that had been fixed. This bug report spreadsheet is included in the **Auxiliary Materials**.

6.2 Debug View

A debug view was created to present the user with useful information regarding game performance and the game state. A screenshot of the debug view can be seen below.

Figure 6.1: Screenshot of debug view.

This includes the FPS (frames per second) being processed in the game engine. This allows the user testing to identify if there are any stages in the game where performance drops. The expected framerate is a consistent 60 FPS; however this can drop when there are a high number of objects being drawn and animated to the screen. The various player and game state variables presented allow the tester to ensure the game is progressing correctly and that there are no inconsistencies throughout the game.

6.3 Test Script

The game has been tested according to the test script found in **Appendix H.** This has been done to verify that the game's main mechanics and features are functioning correctly. Each test script entry includes the name of the test, the description to carry out the test, the expected outcome and whether the test has passed or failed.

6.4 Balance Testing

As the game involved various mechanisms that could dramatically affect the end-result of a player winning or losing, balance testing was carried out to avoid any of the components being ineffective or undesirable. The main focus of balance testing was on gaining points. The game would be considered unbalanced if players were able to win a large amount of points from only certain parts of the game. To do this, 40 sets of detailed player scores were analysed and compared. Each set of results included the total number of points achieved by the player in questions and mini-games. These were checked for any anomalies, such as a player achieving 90% of their points from a single mini-game.

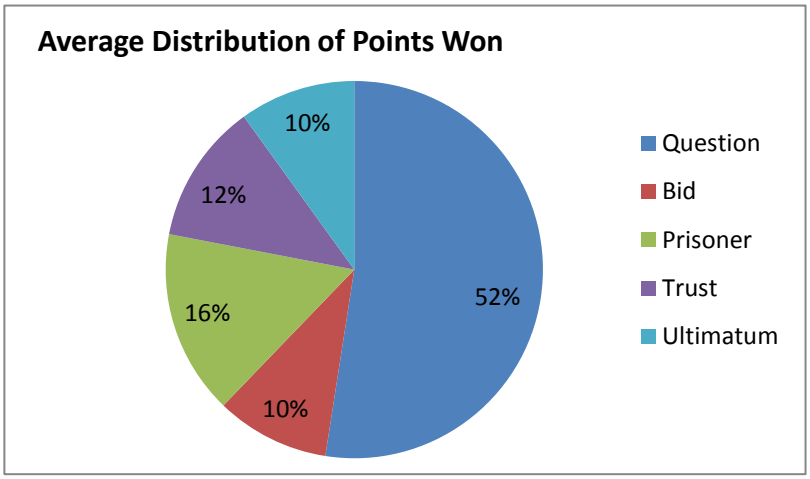


Figure 6.2: Pie chart of average distribution of points won.

The averages of these results were then analysed. **Figure 6.2** demonstrates the average distribution of points won from players. The desired results are for an even distribution across questions and mini-games. On average, questions appear nearly twice as much as mini-games and therefore should distribute twice the amount of points. From these results it can be determined that the game has met the desired results and provides a balanced means of achieving the success criteria: gaining the most points.

Another aspect of balancing is the questions involved in the game. To test this, the sets of results were also checked for the answers given to every question in the game. The answers given for each question were analysed individually ensuring that the percentage of correct answers corresponded to the difficulty of the question. The results were also checked for anomalies, such as questions where all players answered correctly or incorrectly. A summary table of these results can be found in **Appendix I.** A comprehensive document of these tests can be found in the **Auxiliary Materials.**

6.5 End-User Acceptance Testing

A feedback questionnaire was integrated into the game to support end-user acceptance testing. The questionnaire would appear after the player has finished the game (a screenshot is provided in Appendix J). This questionnaire involved 6 questions using a five-point likert scale (strongly agree – strongly disagree) that covered the main aspects of the project.

A set of 40 questionnaire results were analysed to check whether the game meets end-users requirements. The aggregation of these results can be found in **Appendix K.** The following observations were made:

- Only 10% of users did not find the game enjoyable.
- Over 75% of users agree that the game was competitive, engaging and provided accurate and useful feedback.
- All of the users thought they have gained some level of knowledge in rational and logical thinking.
- Over 70% of users found the game easy to understand and follow.
- Finally, all users would consider playing the game again.

The users were also provided with a textbox for any other details they would like to submit. Using the answers the word map below has been created to present the results:

Figure 6.3: Word map created using user comments.

From these results it can be determined that there is a confident response from users that the game is of a satisfactory level.

6.6 Summary

To summarise, various methods of testing were carried out throughout the project, extensive sets of results have been analysed and the game has been evaluated. The game has been considered acceptable, usable and balanced. With bug reporting being used alongside all these tests, it can be said with confidence that the game is relatively bug-free with only a slim chance of non-critical bugs existing of which do not interfere with gameplay.

For this reason, the second project objective has been met: **To create a complete, robust and bug-free game using an iterative model approach to development and testing.**

7 CONCLUSION

Whilst several changes were made throughout the project and a number of problems encountered, the project aim and objectives were successfully met:

The first project objective required a compilation of questions and studies. The project satisfied this objective during the literature survey in **Section 2.1**, selecting 4 studies and a number of questions from reliable sources that were successfully integrated into the game for the evaluation of rational behaviour.

The second objective involved an iterative development approach to create a complete, robust and bug-free game. As discussed in **Section 3.1**, an iterative model of 3 phases (initial, main, final) was used. This provided the flexibility required to meet this objective with requirements, design and testing continuing throughout the project. The test results displayed in **Chapter 5** support this, demonstrating that the game contains no critical bugs and users were satisfied with the final build.

The third objective required the successful implementation of client-server architecture to support network-capability. This has been achieved by designing the network sequences for each component, as discussed in **Chapter 3**, and implementing network protocols that provide considerable control over network communication between the game server and game client. With customisable packet structures and specific packet serialisation/deserialization, complex network sequences were simulated and achieved.

The fourth objective required all information regarding questions to be softcoded, easily manipulated and added with ease. The question system detailed in **Section 4.5** demonstrated powerful functionality that not only excluded all question information from the game code, but also enabled all visual properties of the GUI elements to be softcoded. This was a great achievement as it enabled the creation of a highly scalable game.

All the objectives were met before the project's deadline, the required deliverables were completed and the final build was effectively demonstrated. From this, it can be determined that the project was, overall, a success.

7.1 Reflection

Due to the time constraints of this project some optional requirements were not incorporated into the final build. However, by adopting an iterative approach, it was ensured that all essential requirements were achieved.

A peer-to-peer approach may have been easier to implement, however the additional functionality enabled by the centralised server has proved particularly useful for the game. Research into the various advantages and disadvantages of network models and protocols proved vital to the success of the multiplayer feature.

Throughout the project, unpredictable errors resulted from vigorous testing and as such it was very useful to practice a standardised approach to bug-reporting and test scripts. This provided a centralised state of the game's current stability that I could reference and share with other testers. It was also challenging to ensure the validity of these questions and studies. For this reason, it was important to reference various sources to support any claims made in a question's feedback.

Although the project was successfully completed, I believe several changes would have produced better results, the most important being the approach to the software development process. The iterative model adopted helped achieve the project objectives; however with the amount of testing required a spiral approach would have been more efficient. This is due to the spiral model enabling the creation of prototypes of the game after each phase; this would have permitted end-user testing from the beginning of the project, rather than the final phase using the iterative model.

Another important change that I would consider would be to separate the different tiles into either questions or mini-games, excluding difficulty levels. This would have provided an interesting competitive environment that may have been more enjoyable as players would be participating in the same question, regardless of difficulty.

7.2 Future Development

There is still room for improvement and extra functionality due to the flexibility achieved from creating the network, board game, feedback and question systems. In particular, the content involved in the game can be extended. Each individual component has been created with customisation and further development in mind. For instance, the question system facilitates simple integration of new question descriptions and feedback with all other systems updating accordingly.

As discussed in **Section 4.7.2**, the optional requirement of a Java applet was not successfully completed. Due to the difficulty in integrating the Kryonet network library and the time available this was not considered as a priority but will be considered as an important feature of deployment in future development.

Another important feature that will be considered in future development is a centralised 'Sever Lobby' function. The network system created currently stores all connections and has the appropriate functionality in order to send messages to all connected clients and display the game sessions currently being hosted. With this a lobby can be simulated to allow all the clients that are connected to communicate through an instant messaging system (similar to the one already implemented) and join each other's games publicly.

7.3 Personal Development

Prior to this project I had no experience of video game design or development, encountering numerous challenges throughout the implementation phase. However, after completing the final build of the game, it has become an area of personal interest. By adopting an iterative approach, I have gained experience in the various aspects involved in the software development process, such as requirements analysis and testing. In order to complete this project it was necessary to learn a number of different technologies, including the Slick2D game engine, Kryonet network library and TWL GUI library. Alongside this, I was able to utilise HTML, CSS and XML in conjunction with Java to develop unique solutions for separating game content from game code. From creating art assets to researching software licenses, a number of new skills were required to complete this project, resulting in a great deal of personal development.

8 WORKS CITED

[1] **Kahneman, Daniel.** The Lazy Controller. *Thinking, Fast and Slow.* s.l. : Penguin Group, 2011, p. 44.

[2] **Kahneman, Daniel.** The Lazy Controller. *Thinking, Fast and Slow.* s.l. : Penguin Group, 2011, p. 49.

[3] **Friedman, Milton.** *Essays in Positive Economics.* s.l. : Phoenix Books, 1953. pp. 15, 22.

[4] **Friedman, Milton.** The Methodology of Positive Economics. *Essays in Positive Economics.* 1953, pp. 15, 22, 31.

[5] **Mazie, Steven.** How Rational Are You? Try This Quiz. *Big Think.* [Online] June 2012. http://bigthink.com/praxis/how-rational-are-you-try-this-quiz.

[6] **HelloQuizzy.** How Rational Are You Really Test. [Online] 2012. http://helloquizzy.okcupid.com/tests/the-how-rational-are-you-really-test.

[7] **Kleiner, Kurt.** How Rational Are You? *University of Toronto Magazine.* [Online] August 2009. http://www.magazine.utoronto.ca/summer-2009/rationality-quiz/.

[8] **Vanity Fair.** The Quiz Daniel Kahneman Wants You to Fail. *Vanity Fair.* [Online] December 2011. http://www.vanityfair.com/business/features/2011/12/kahneman-quiz-201112.

[9] **Jubert, Tom.** ir/rational Redux. *Tom Jubert: Writing Portfolio.* [Online] July 2012. http://www.tomjubert.com/.

[10] **Jubert, Tom**. ir/rational Redux. *Newgrounds.* [Online] September 2012. http://www.newgrounds.com/portal/view/598731.

[11] **Gintis, Herbert.** Prisoner's Dilemma. *The Bounds of Reason: Game Theory and the Unification of the Behavioral Sciences.* s.l. : Princeton University Press , 2009.

[12] **Gintis, Herbert.**. Trust Game. *The Bounds of Reason: Game Theory and the Unification of the Behavioral Sciences.* s.l. : Princeton University Press, 2009, p. 71.

[13] **Milgrom, Paul R.** First-Price Sealed-bid Auction. *A theory of auctions and competitive bidding.* s.l. : Institute for Mathematical Studies in the Social Sciences, 1981.

[14] **Gintis, Herbert.** Ultimatum Game. *The Bounds of Reason: Game Theory and the Unification of the Behavioral Sciences .* s.l. : Princeton University Press, 2009.

[15] **Kahneman, Daniel.** *Thinking, fast and slow.* s.l. : Penguin Group, 2011.

[16] **Kahneman, Daniel.** *Judgment under Uncertainty: Heuristics and Biases.* s.l. : Cambridge University Press, 1982.

[17] **Gintis, Herbert.** *The Bounds of Reason: Game Theory and the Unification of the Behavioral Sciences.* s.l. : Princeton University Press, 2009.

[18] **Smullyan, Raymond M.** *Forever Undecided: A Puzzle Guide to Godel.* s.l. : Oxford Paperbacks, 2000.

[19] **Weirich, Paul.** *Equilibrium and Rationality: Game Theory Revised by Decision Rules.* s.l. : Cambridge University Press, 2007.

[20] **Microsoft.** Microsoft XNA Framework Redistributable 4.0. *Microsoft - Official Website.* [Online] September 2010. http://www.microsoft.com/en-gb/download/details.aspx?id=20914.

[21] **Microsoft.** Microsoft: Next Generation of Games Starts With XNA. *Microsoft News Center.* [Online] March 2004. https://www.microsoft.com/en-us/news/press/2004/mar04/03-24xnalaunchpr.aspx.

[22] **Microsoft.** Microsoft Download Center. *Microsoft - Official Website.* [Online] November 2008. http://www.microsoft.com/en-us/download/details.aspx?id=12460.

[23] **Klucher, Michael.** XNA Framework Networking and Live Requirements. *Microsoft Development Blogs.* [Online] November 2007. http://blogs.msdn.com/b/xna/archive/2007/11/16/xna-framework-networking-and-live-requirements.aspx.

[24] **Team, IONSTAR Studios Developer.** Squid Overview. *IONSTAR Studios.* [Online] March 2012. http://www.ionstar.org/?page_id=4).

[25] **Team, XNA Simple GUI.** XNA Simple GUI 2.0 Overview. *XNA Simple GUI.* [Online] July 2010. http://simplegui.codeplex.com/.

[26] **Team, Pygame.** Pygame - Homepage. *Pygame - Official Website.* [Online] June 2012. http://www.pygame.org/news.html.

[27] **Shinners, Pete.** Python Pygame Introduction. *Pygame - Official Website.* [Online] October 2009. http://www.pygame.org/docs/tut/intro/intro.html.

[28] **Pygame.** GNU LESSER GENERAL PUBLIC LICENSE. *Pygame - Official Website.* [Online] February 1999. http://www.pygame.org/LGPL.

[29] **Pygame.** About Pygame. *Pygame - Official Website.* [Online] August 2009. http://www.pygame.org/wiki/about.

[30] **Shinners, Pete.** Pygame - Resources. *Pygame - Official Website.* [Online] March 2013. http://www.pygame.org/wiki/resources.

[31] **Krause, Michael.** hotswap 0.1. *Python - Official Website.* [Online] 2010. https://pypi.python.org/pypi/hotswap/0.1.

[32] **TwistedMatrixLabs.** What is Twisted? *Twisted - Official Website.* [Online] April 2013. http://twistedmatrix.com/.

[33] **TwistedMatrixLabs.** The Twisted Advantage. *Twisted - Official Website.* [Online] April 2012. http://twistedmatrix.com/trac/wiki/TwistedAdvantage.

[34] **Rogers, Peter.** PGU - Phil's pyGame Utilities. *Pygame - Official Website.* [Online] March 2012. https://code.google.com/p/pgu/.

[35] **Appen, Marcus von.** OcempGUI - a GUI Engine for Pygame and more. *Ocean Empire.* [Online] May 2008. http://ocemp.sourceforge.net/gui.html.

[36] **Team, Slick2D.** Slick2D - Homepage. *Slick2D - Official Website.* [Online] October 2012. http://www.slick2d.org/.

[37] **Rychlik-Prince, Caspian.** About LWJGL. *LWJGL - Official Website.* [Online] December 2012. http://www.lwjgl.org/wiki/index.php?title=About_LWJGL.

[38] **Team, Slick2D.** Slick2D Wiki. *Slick2D - Official Website.* [Online] October 2012. http://www.slick2d.org/wiki/index.php/Main_Page.

[39] **Oracle.** Java Network Launch Protocol (JNLP) Support. *Oracle Technology Network.* [Online] http://www.oracle.com/technetwork/java/javase/index-142562.html.

[40] **Sweet, Nathan.** kryonet - TCP and UDP Client/Server Library for Java. *Kryonet - Official Website.* [Online] September 2012. https://code.google.com/p/kryonet/.

[41] **Oracle.** Java SE Documentation. *Swing (JavaTM Foundation Classes).* [Online] 2011. http://docs.oracle.com/javase/6/docs/technotes/guides/swing/.

[42] **Mann, Matthias.** TWL - Themable Widget Library. *What is TWL?* [Online] April 2013. http://l33tlabs.org/.

[43] **Gafferon Games.** Networking. *Gafferon Games.* [Online] October 2008. http://gafferongames.com/networking-for-game-programmers/.

[44] **Scott Coleman, Jay Cotton.** DOOM on the Internet via TCP/IP. *FAQs.* [Online] November 1995. http://www.faqs.org/faqs/games/doom/howto-tcp/#ixzz0daOnpBYn.

[45] **Red Key Blue Key.** Pirnciples of Good Game Design. *Red Key Blue Key - Offiicial Website.* [Online] September 2011. http://www.redkeybluekey.com/2011/09/8-principles-of-good-game-design.html.

[46] **Salen, Katie and Zimmerman, Eric.** *Rules of Play: Game Design Fundamentals.* s.l. : MIT Press, 2003.

[47] **Larman, Craig.** Iterative Development Illustration. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* s.l. : Prentice Hall, 2004.

[48] **Wiegers, Karl E.** First Things First: Prioritizing Requirements. *Process Impact.* [Online] 1999. http://www.tarrani.net/linda/prioritizing.pdf.

[49] **OpenGameArt.** Open Game Art Homepage. *OpenGameArt.* [Online] April 2013. http://opengameart.org/.

[50] **Cambridge, University of.** Rationality. *The Cambridge Dictionary of Philosphy.* 1999, p. 772.

[51] **Microsoft.** Microsoft Software License Terms - Microsoft XNA Game Studio. *Microsoft.* [Online] http://xbox.create.msdn.com/downloads/?id=166&lc=1033.

[52] **Microsoft.** Microsoft XNA Frequently Asked Questions. *Microsoft.* [Online] October 2011. http://msdn.microsoft.com/en-us/xna/aa937793.aspx.

[53] **Crossley, Rob.** Microsoft email confirms plan to cease XNA support. *CVG UK.* [Online] February 2013. http://www.computerandvideogames.com/389018/microsoft-email-confirms-plan-to-cease-xna-support/#.

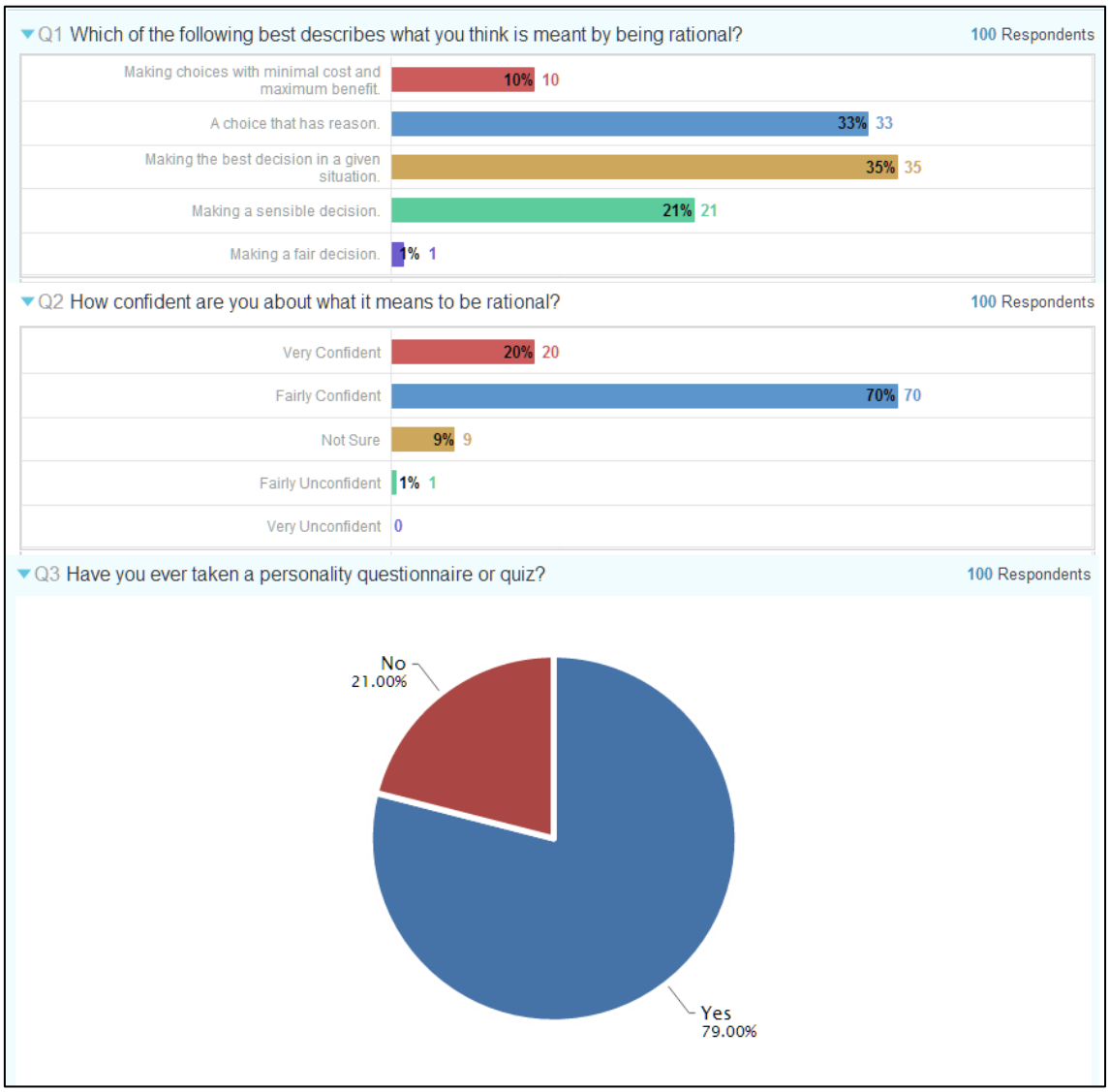
[54] **Library, Lightweight Game.** LWJGL - License. *Lightweight Java Game Library - Official Website.* [Online] 2002. http://www.lwjgl.org/license.php.

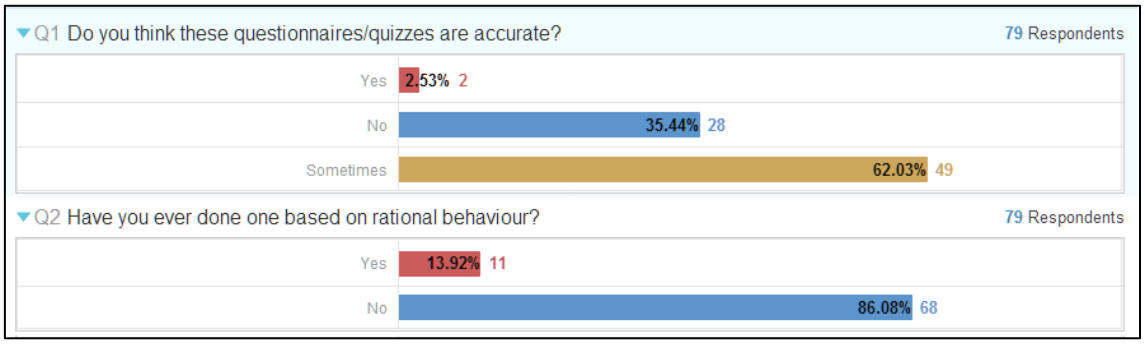
[55] **Hicks, Matt.** Java Gaming Network - Project Home. *Java Gaming Network - Official Website.* [Online] July 2011. https://code.google.com/p/jgn/.

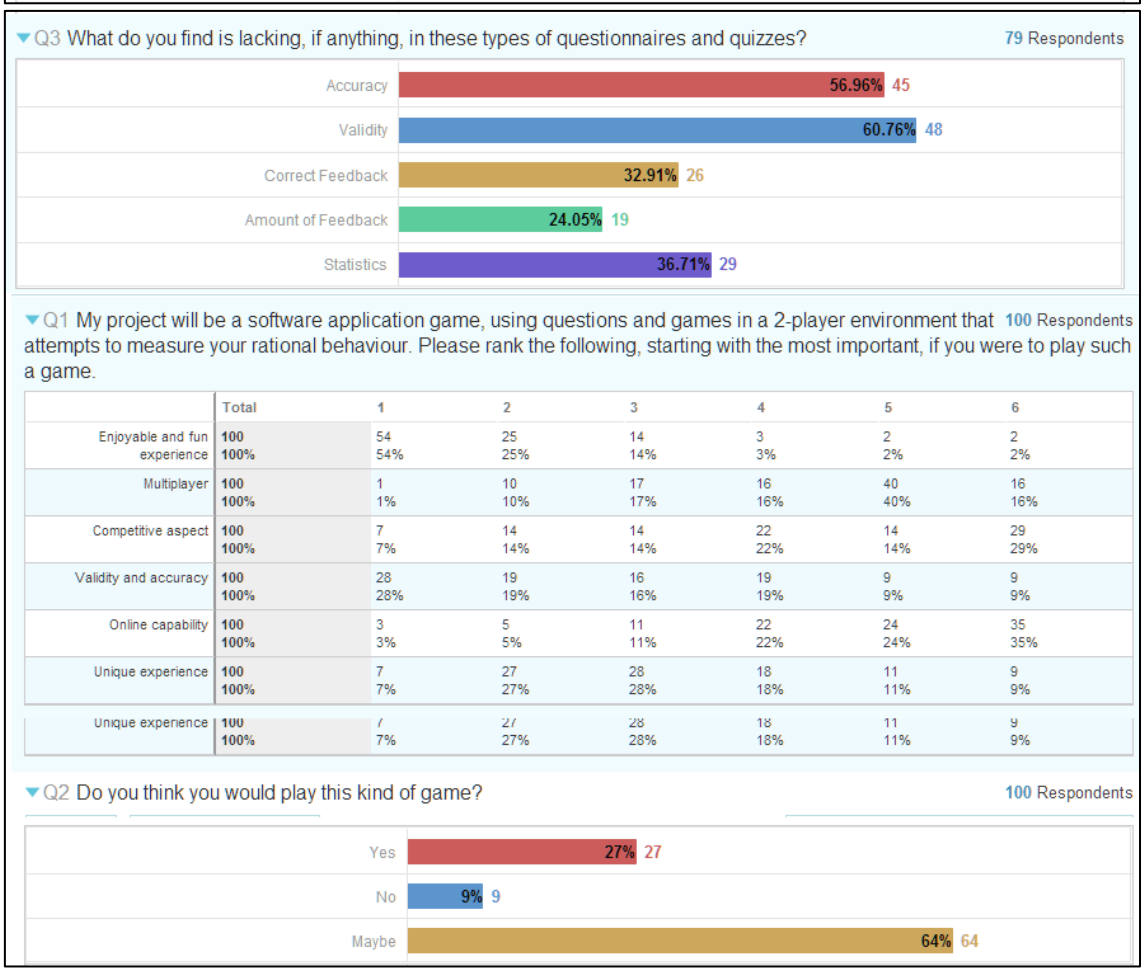
9 APPENDIX

9.1 Appendix A – Initial Audience Survey

100 Respondents, aged 16-26 in the UK, 70% Male 30% Female. 6-10 questions.







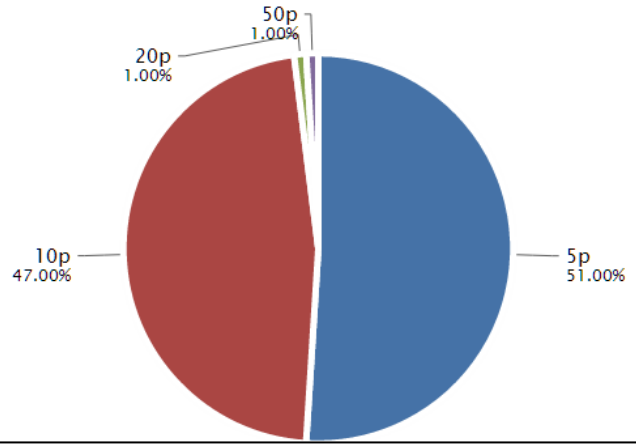
Q1 My project will be a software application game, using questions and games in a 2-player environment that attempts to measure your rational behaviour. Please rank the following, starting with the most important, if you were to play such a game. 100 Respondents

	Total	1	2	3	4	5	6
Enjoyable and fun experience	100 100%	54 54%	25 25%	14 14%	3 3%	2 2%	2 2%
Multiplayer	100 100%	1 1%	10 10%	17 17%	16 16%	40 40%	16 16%
Competitive aspect	100 100%	7 7%	14 14%	14 14%	22 22%	14 14%	29 29%
Validity and accuracy	100 100%	28 28%	19 19%	16 16%	19 19%	9 9%	9 9%
Online capability	100 100%	3 3%	5 5%	11 11%	22 22%	24 24%	35 35%
Unique experience	100 100%	7 7%	27 27%	28 28%	18 18%	11 11%	9 9%
Unique experience	100 100%	7 7%	27 27%	28 28%	18 18%	11 11%	9 9%

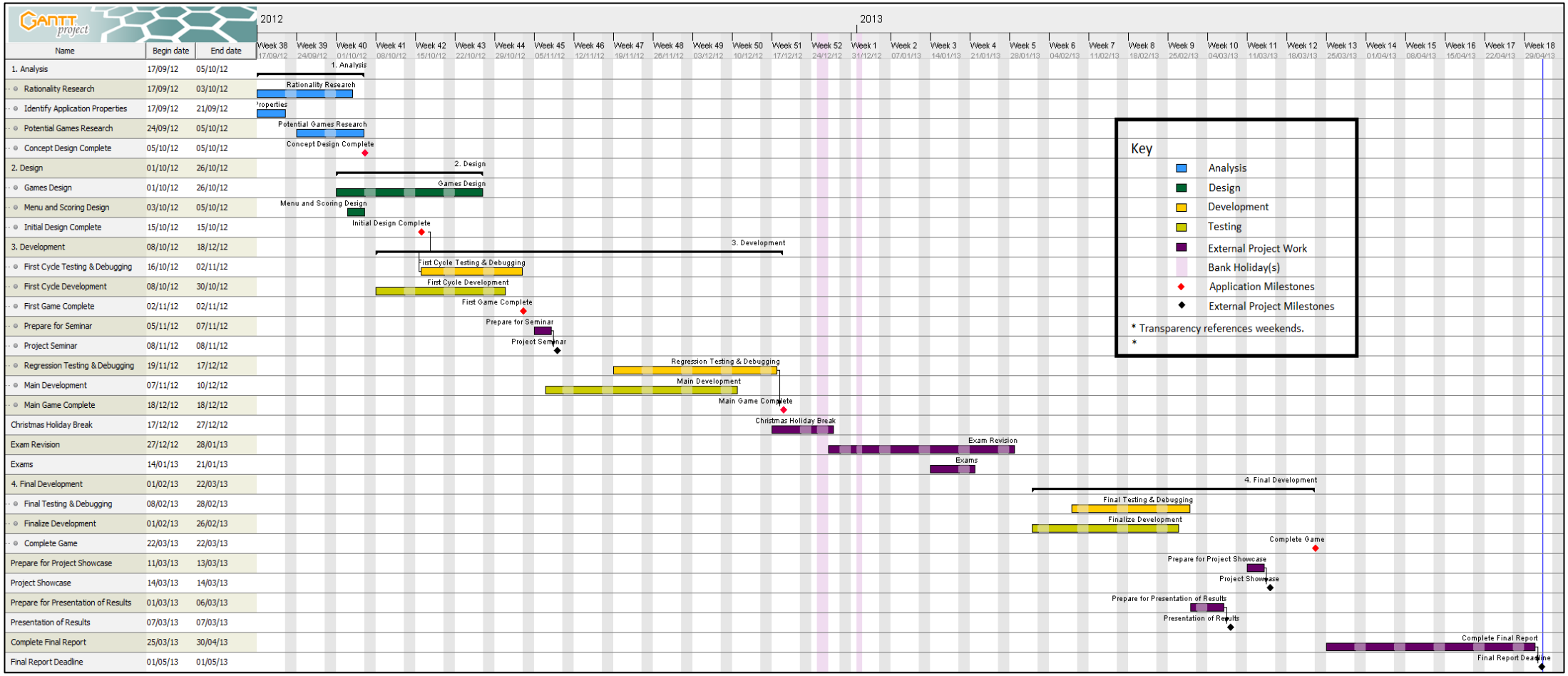
Q3 Please rank the following in order of what you think is most important in a game, starting with the most important. 100 Respondents

	Total	1	2	3	4	5	6
Good User Interface	100 100%	9 9%	18 18%	35 35%	14 14%	12 12%	12 12%
Good Graphics	100 100%	5 5%	3 3%	8 8%	22 22%	18 18%	44 44%
Good Gameplay	100 100%	40 40%	28 28%	12 12%	8 8%	9 9%	3 3%
Fun	100 100%	32 32%	32 32%	17 17%	13 13%	4 4%	2 2%
Simple and Easy Controls	100 100%	4 4%	11 11%	19 19%	19 19%	37 37%	10 10%
Simple and Easy Instructions	100 100%	10 10%	8 8%	9 9%	24 24%	20 20%	29 29%

Q4 Finally, please answer the following question: A bat and ball cost £1.10. The bat costs £1 more than the ball. How much does the ball cost? 100 Respondent



9.2 Appendix B - Project Gantt Chart



9.3 Appendix C – Detailed Functional Requirements

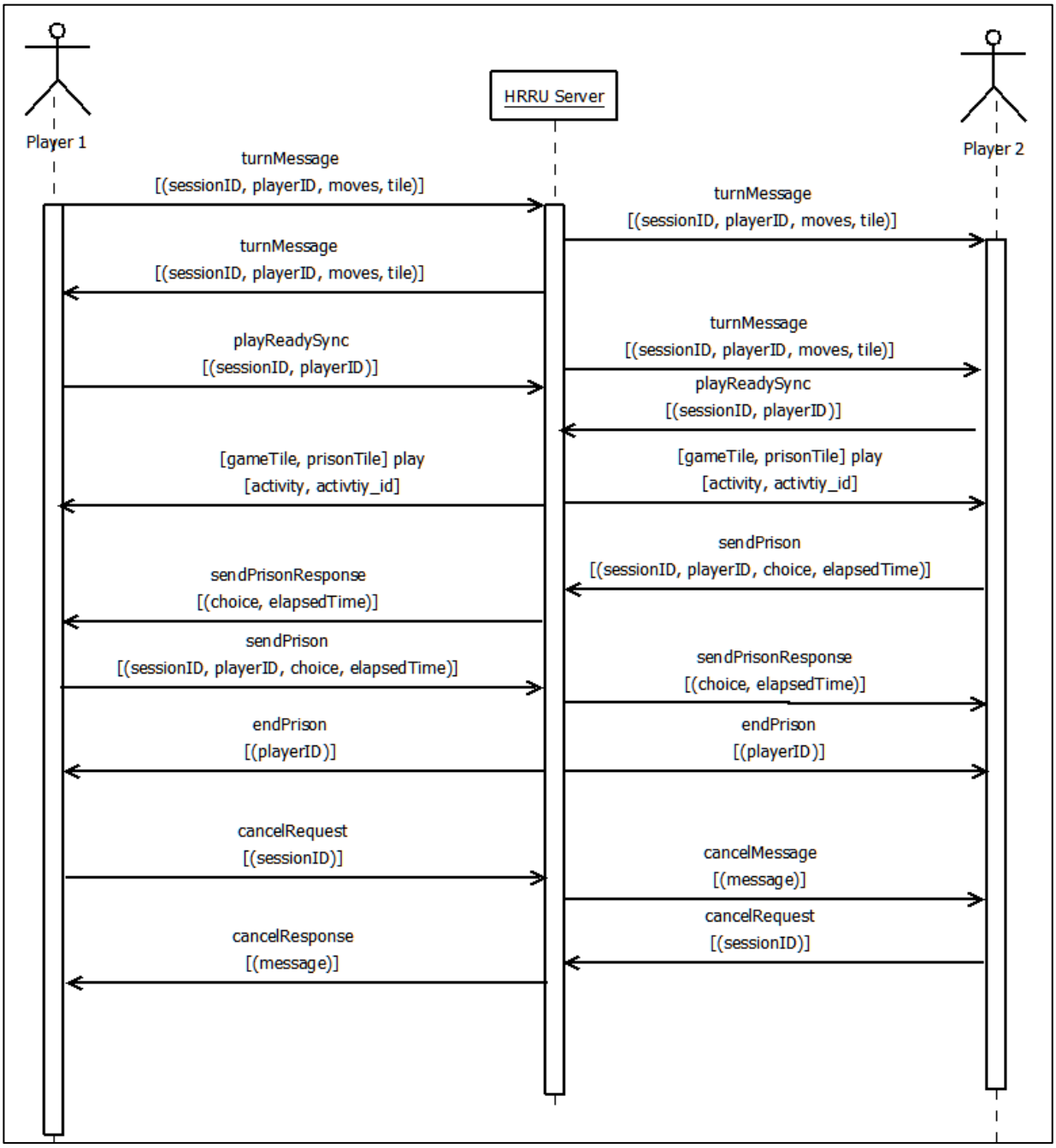
Reference #	Requirement Description	Priority
Main Game Screen		
MFR1	A 9 by 9 square board comprised of 4 different tiles (easy, medium, hard and minigame) will be displayed.	**Essential**
MFR2	The game timer will be displayed in a large 12pt font in the corner of the screen.	**Essential**
MFR3	An IM (Instant Messaging) chat box will be displayed in the corner of the screen.	**Conditional**
MFR4	Players will be able to send and receive messages using the IM chat box.	**Conditional**
MFR5	The player's avatar will be displayed at the correct tile on the board.	**Essential**
MFR6	Players will be notified when it is their turn.	**Essential**
MFR7	Players will be able to click a button to roll the dice when it is their turn.	**Essential**
MFR8	The dice will generate a pseudorandom value between 1 and 6.	**Essential**
MFR9	A pseudorandom animation of the dice will be displayed simulating a player's roll.	**Optional**
MFR10	The player's position will be updated on each player's screen after the completion of a turn.	**Essential**
MFR11	The game will enter the minigame screen if either player is on a minigame tile after both players have completed their turn.	**Essential**
MFR12	The game will enter the question screen if both players are on a question tile after both players have completed their turn.	**Essential**
MFR13	Players will receive the same question if they land on the same difficulty question tile.	**Essential**
MFR14	The game will cycle through the minigames fairly so that they all appear evenly.	**Essential**
Question Screen		
QFR1	Each multiple choice question will have a description and a list of clickable buttons representing the choices.	**Essential**
QFR2	Players will be prompted to confirm their answer after selecting a choice.	**Conditional**
QFR3	The game client will store the question ID, the difficulty level and the answer chosen and the time elapsed when a choice has been confirmed.	**Conditional**
QFR4	The game client will store and start a countdown timer from 120 seconds once a question has started.	**Essential**
QFR5	The question timer will be displayed in a large 12pt font in the corner of the screen.	**Essential**
QFR6	The game will display a notification to both players when either player has finalised a choice.	**Conditional**
QFR7	The game will display the results once the player's game client timer reaches 0.	**Essential**
QFR8	The game will display the results once both players have made a choice.	**Essential**
QFR9	The results will include each player's outcomes: whether they have gotten the question correct, the difficulty level of their question, the time taken to answer the question and the player's new score.	**Conditional**
QFR10	The game will use the difficulty level and time remaining to update the player's score if a correct answer is given.	**Conditional**
QFR11	The game will return to the main game screen after the results have been displayed for 10 seconds.	**Essential**
General Minigame		
GMFR1	The game will display an explanation of the mini game.	**Essential**
GMFR2	Players will be prompted to confirm their answer after selecting any choices.	**Conditional**
GMFR3	The game will display the results once the player's game client timer reaches 0.	**Essential**
GMFR4	The game will display the results once both players have made a choice.	**Essential**
GMFR5	The minigame timer will be displayed in a large 12pt font in the corner of the screen.	**Essential**
GMFR6	The game will display a notification to both players when either player has finalised a choice.	**Conditional**

Prisoner Minigame Screen		
PFR1	Players can click one of two buttons to “cooperate” or “betray”.	**Essential**
PFR2	The game will default to the choice to “betray” if no selection is made.	**Essential**
PFR3	The game client will store and start a countdown timer from 50 seconds once the minigame has started.	**Essential**
PFR4	When both players “cooperate”, the game will update their scores by 150.	**Essential**
PFR5	When the player’s choices differ, the game will update the player’s score who chose to “cooperate” by 50 and the player’s score who chose to “betray” by 250.	**Essential**
PFR6	The results will include: each player’s choice, the points gained and their new score.	**Conditional**
PFR7	The game will return to the main game screen after the results have been displayed for 8 seconds.	**Essential**
PFR8	The game client will store the answer chosen and the time elapsed when a choice has been confirmed.	**Conditional**
Sealed Bidding Minigame Screen		
SFR1	The game will default to a bid of “0” if no selection is made.	**Essential**
SFR2	The game client will store and start a countdown timer from 60 seconds once the minigame has started.	**Essential**
SFR3	The results will include: each player’s bid and their new score.	**Conditional**
SFR4	The game will generate a pseudorandom “item” including an item name, item description and value.	**Optional**
SFR5	Players can enter their bid by typing a number into a text box.	**Essential**
SFR6	The bidding text box will only accept numeric values.	**Essential**
SFR7	The game will not accept bids less than 0 or more than the player’s current available points.	**Essential**
SFR8	The game will update the score of the winner by the value of the item subtracted by their bid.	**Essential**
Trust Minigame Screen		
TFR1	The game will default to the value “0” if no value is finalised.	**Essential**
TFR2	The game client will store and start a countdown timer from 80 seconds once the minigame has started.	**Essential**
TFR3	The game will generate a pseudorandom total give value from 20 to 100 in multiples of 20.	**Conditional**
TFR4	The game will generate a pseudorandom multiplier between and including 2 and 5.	**Conditional**
TFR5	The game will select one player to “give” and one player to “return” interchangeably every trust minigame.	**Essential**
TFR6	The “give” player can enter the give value by typing into a text box.	**Essential**
TFR7	The “return” player can enter the return value by typing into a text box.	**Essential**
TFR8	The text box will only accept numeric values.	**Essential**
TFR9	The game will not accept a give value less than 0 or more than the total give value.	**Essential**
TFR10	The game will not accept a return value less than 0 or more than the total return value.	**Essential**
TFR11	The game will update the score of both players using the values entered.	**Essential**
Ultimatum Minigame Screen		
UFR1	The game client will store and start a countdown timer from 100 seconds once the minigame has started.	**Essential**
UFR2	The game will generate a pseudorandom total value from 50 to 250 in multiples of 50.	**Conditional**
UFR3	The game will select one player to “propose” and one player to “decide” interchangeably every ultimatum minigame.	**Essential**

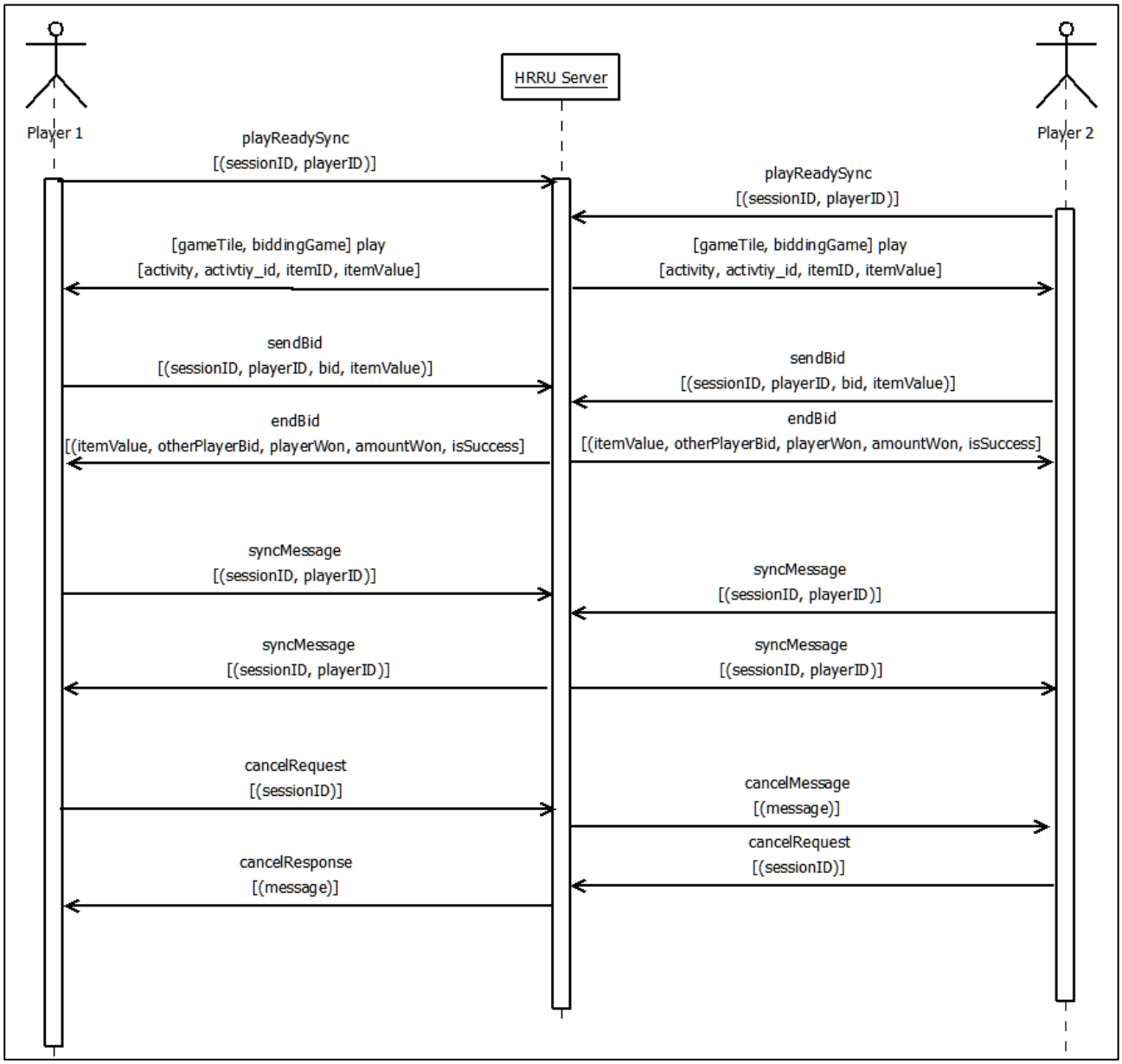
UFR4	The “propose” player can enter the proposed value by typing into a box.	**Essential**
UFR5	The “decide” player can accept or reject the proposed value by clicking the accept or reject buttons.	**Essential**
UFR6	The text box will only accept numeric values.	**Essential**
UFR7	The game will not accept a propose value less than 0 or more than the total value.	**Essential**
UFR8	The game will default to a proposed value of “0” if no proposed value is entered by the “propose” player.	**Essential**
UFR9	The game will default to “reject” if neither accept nor reject button is clicked by the “decide” player.	**Essential**
UFR10	The game will update both players’ scores by the proposed values if the “decide” player accepts.	**Essential**
UFR11	The game client will store and start a countdown timer from 60 seconds once the minigame has started.	**Essential**
End Game Screen		
EFR1	The game will calculate and display the top 2 achievements the player has completed during the game session using images.	**Conditional**
EFR2	The game will display a “winner” image if the player has achieved the higher score.	**Essential**
EFR3	The game will display a “loser” image if the player has achieved the lower score.	**Essential**
EFR4	The game will display an image for the rank achieved, from A to E, based on the player’s score.	**Conditional**
EFR5	The game will display a paragraph summarising the player’s results.	**Essential**
EFR6	The game will display statistics on each question and minigame played.	**Essential**
EFR7	The game will display feedback on each question played.	**Essential**
EFR8	The game client will send the results of each minigame, question and the overall score of each player to the game server.	**Conditional**
EFR9	The player will be able to navigate through the statistics and feedback of each minigame and question through labelled buttons.	**Essential**
EFR10	The game will prompt the player to complete feedback questionnaire.	**Optional**

9.4 Appendix D – Minigame Network Sequence Diagrams

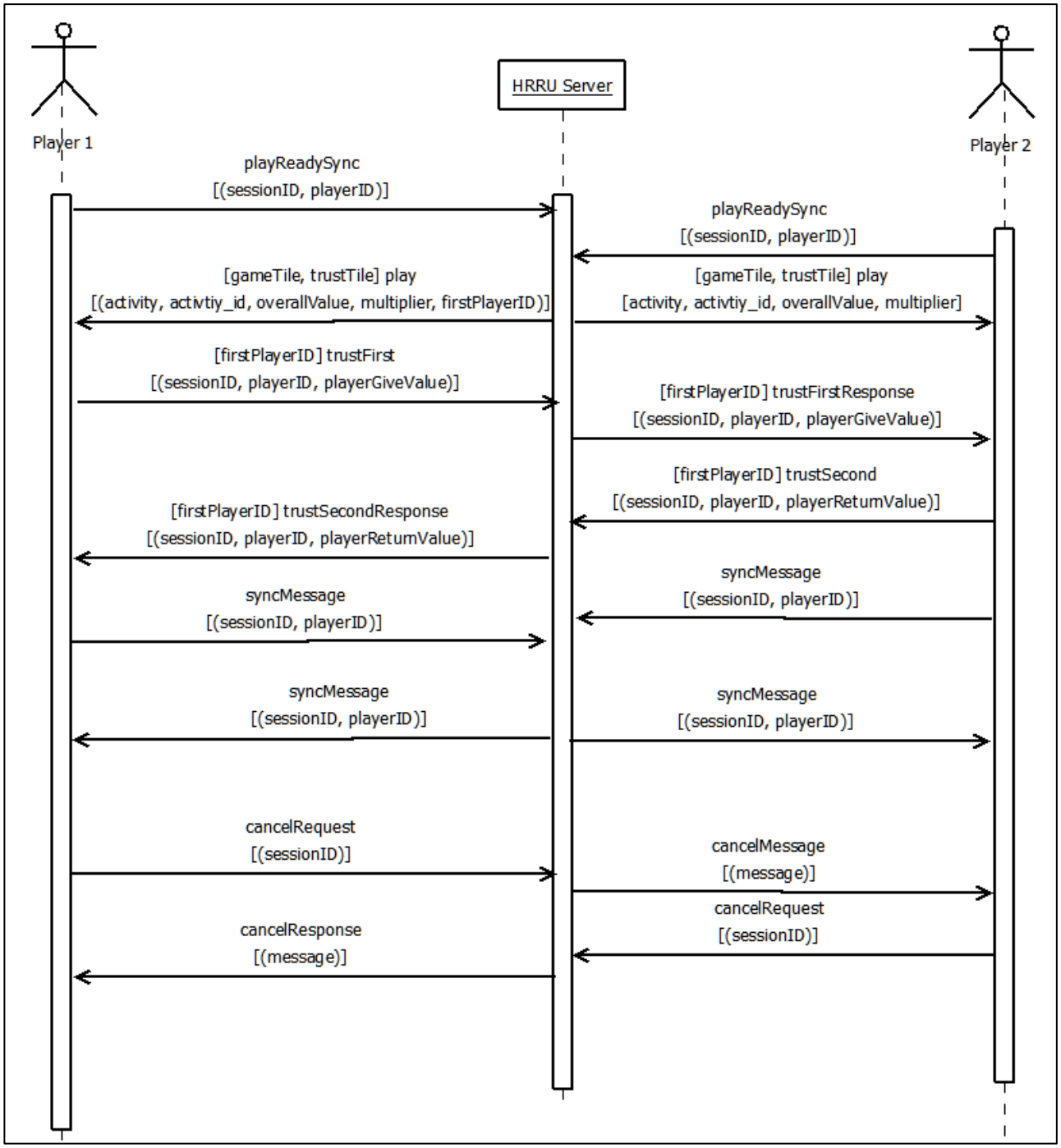
Prisoner's Dilemma Minigame:



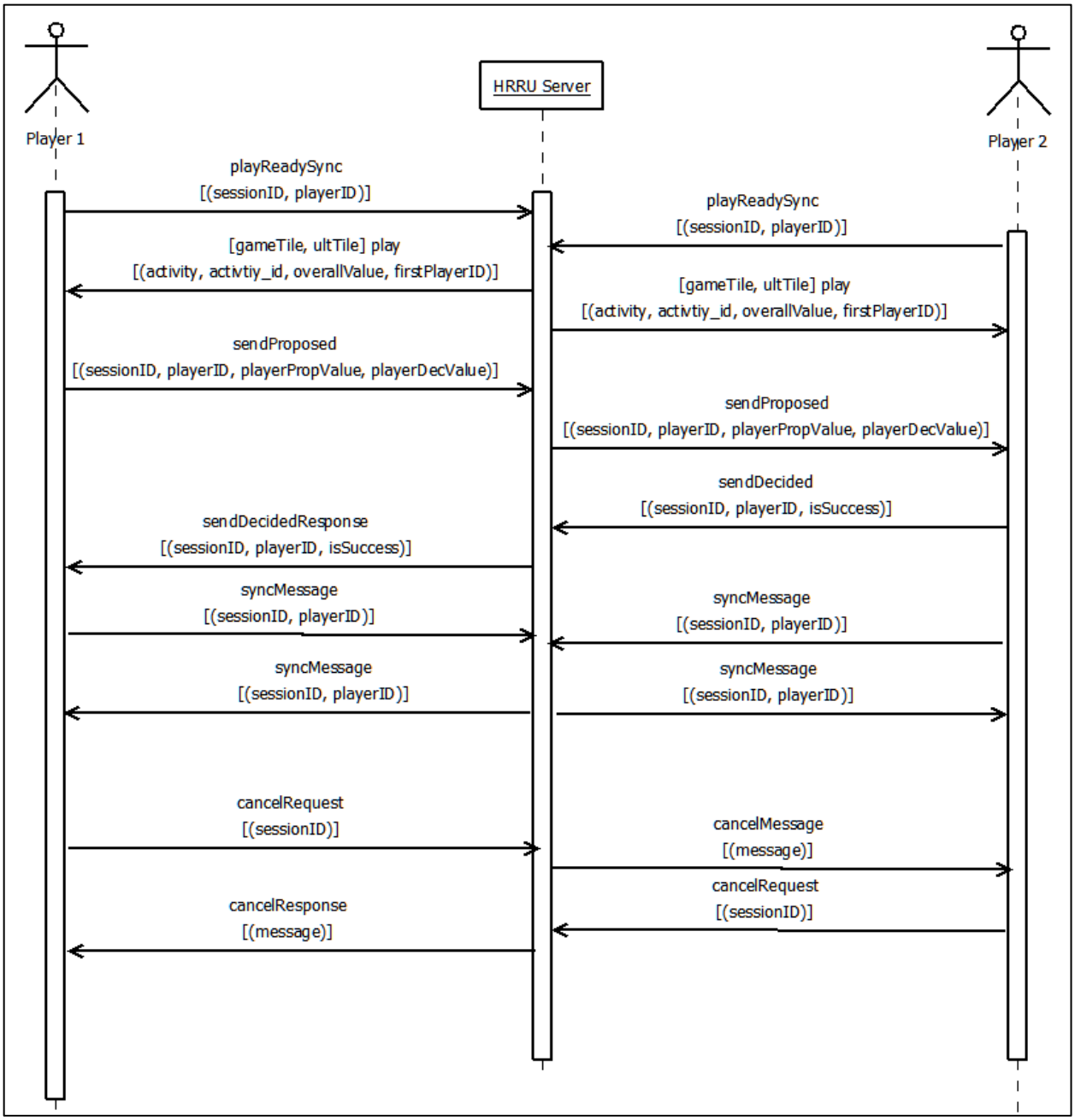
Sealed Bidding Auction Minigame:



Trust Minigame:



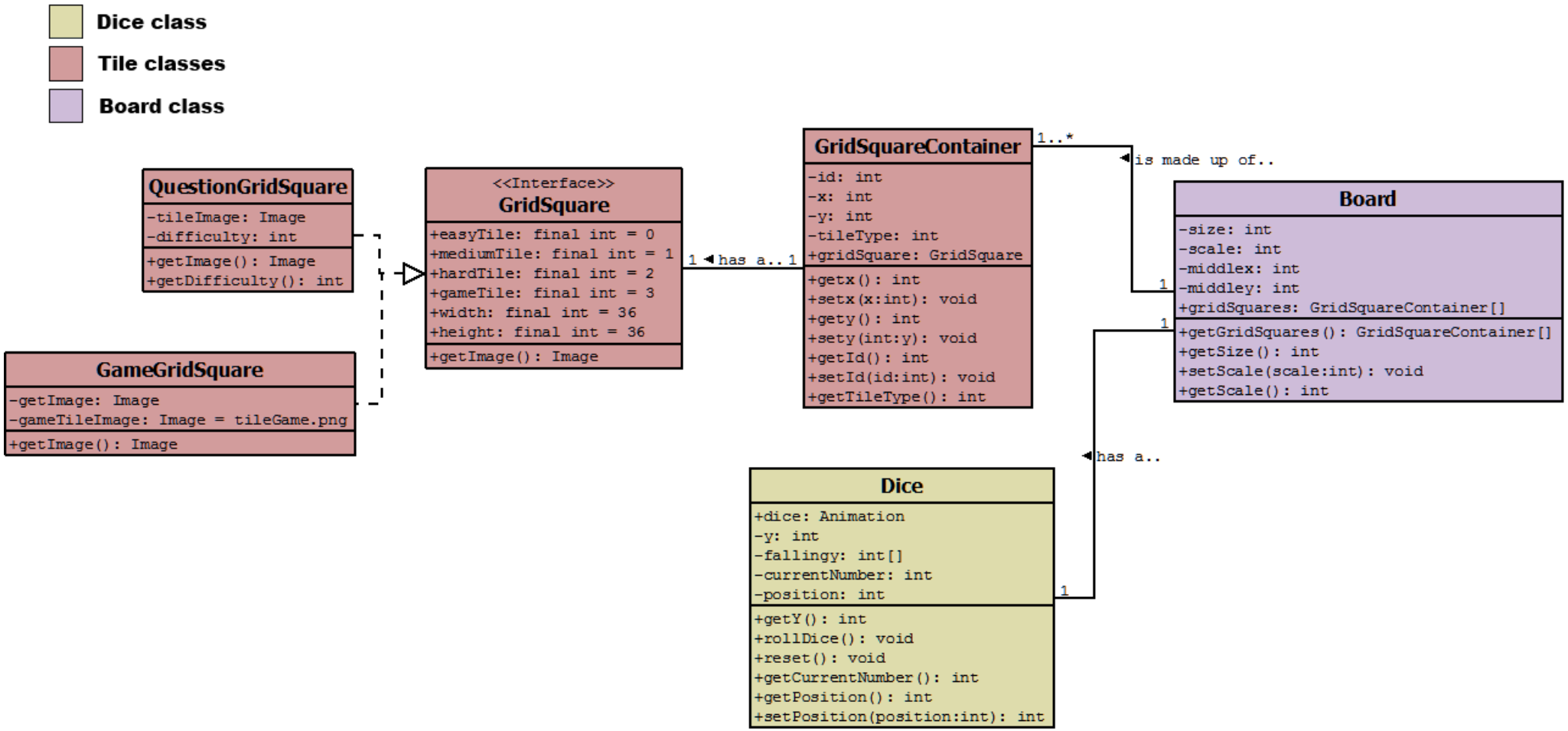
Ultimatum Minigame:



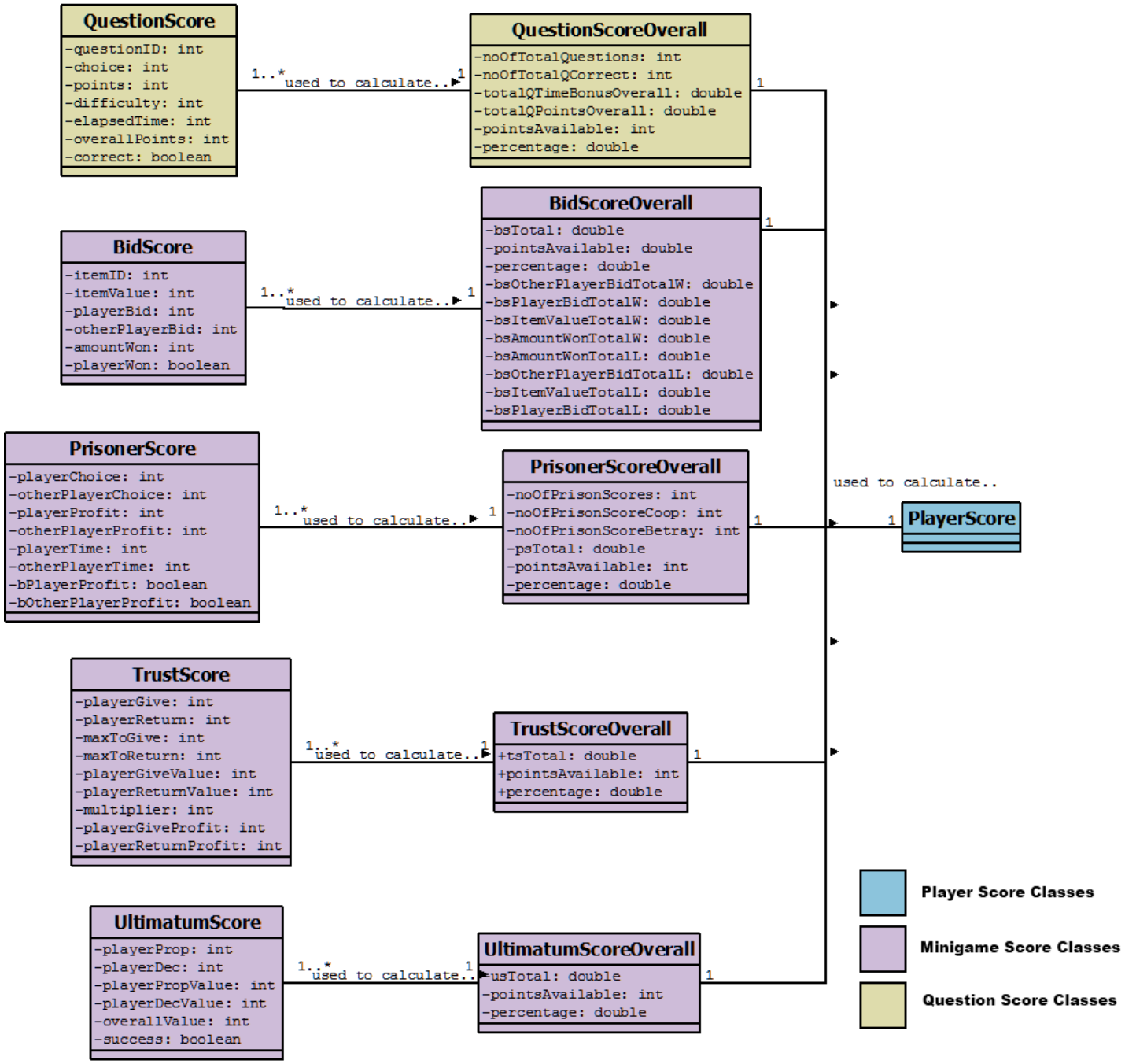
9.5 Appendix E - Packet Structures

```
Packet00SyncMessage { int sessionID; int player; }
Packet0CreateRequest { String player1Name; String password; }
Packet1CreateAnswer { boolean accepted = false; int sessionID; String password; }
Packet2JoinRequest { String player2Name; int sessionID; String password; }
Packet3JoinAnswer { String player1Name; boolean accepted = false; int sessionID; String password; }
Packet4ConnectionEstablished { String player2Name; }
Packet5CancelRequest { int sessionID; }
Packet6CancelRequestResponse { String message; }
Packet7Ready { int sessionID; int player; }
Packet8Start { int sessionID; int[] board; }
Packet9CharacterSelect { int sessionID;  int player; int characterID; }
Packet10ChatMessage { int playerID; String message; }
Packet11TurnMessage { int sessionID; int playerID; int moves; int tile; }
Packet12PlayReady { int sessionID; int player;}
Packet13Play { int activity; int activity_id; int secondary_id; int secondary_value; int third_value; }
Packet14QuestionComplete { int sessionID; int player; int choice; int points; int difficulty; int elapsedtime; int overall; boolean correct; }
Packet15PuzzleComplete { int sessionID; int player; int points; int difficulty; int elapsedtime; int overall; boolean correct; }
Packet16SendBid { int sessionID; int player; int bid; int itemValue; }
Packet17EndBid { int itemValue; int otherPlayerBid; int playerWon; int amountWon; boolean win;}
Packet18TrustFirst { int sessionID; int player; int playerGiveValue; }
Packet19TrustSecond { int sessionID; int player; int playerReturnValue; }
Packet20SendPrison { int sessionID; int player; int choice; int elapsedTime; }
Packet21EndPrison { }
Packet22PropUlt { int sessionID; int player; int playerPropValue; int playerDecValue; }
Packet23DecUlt { int sessionID; int player; boolean success; }
Packet24SendScore { int sessionID; int player; String name; int score; }
Packet25AllScores { String[] names; int[] scores; }
Packet26Feedback { int[] feedback; String otherComments; }
Packet27QuestionAnswers { int[] answers; int questionScore; int bidScore; int prisonScore; int trustScore; int ultScore; int playerScore; }
```

9.6 Appendix F - Board System Full Class Diagram



9.7 Appendix G - Results and Scores Class Diagram



9.8 Appendix H – Test Scripts

Test Name	Test Description	Expected Outcome	Actual Outcome
Host Game	Host a game with player name "Alice" and password "1234".	The game should present the sessionID, password and notify the player that they are now waiting for a second player.	PASS
Join Game	Join the game created above with player name “Bob”.	The game should connect to the session above and present both player names “Alice” and “Bob”	PASS
Character Select	Select a character and finalise the choice.	The game should update the player’s avatar to the selected character.	PASS
Send Message	Send a “Hello” message using the chatbox.	The game should update the chatbox for both players’ with the player’s name followed by the message “Hello”.	PASS
Roll Dice	Click the button to roll the dice.	The game should present a pseudorandom dice animation and generate a pseudorandom number between 1 and 6. It should then update the player’s position on the board by this amount.	PASS
Both Easy Question	Complete 2 player turns with the outcome of landing on 2 easy question tiles.	The game should present the same question with an easy difficulty to both players.	PASS
Both Medium Question	Complete 2 player turns with the outcome of landing on 2 medium question tiles.	The game should present the same question with a medium difficulty to both players.	PASS
Both Hard Question	Complete 2 player turns with the outcome of landing on 2 hard question tiles.	The game should present the same question with a hard difficulty to both players.	PASS
Different Question	Complete 2 player turns with the outcome of landing on 2 different difficulty questions.	The game should present a question to each player depending on the difficulty tile.	PASS
Minigame	Complete 2 player turns with the outcome of at least 1 player landing on a minigame tile.	The game should present the same minigame to both players.	PASS
Both Player’s Question Answered	Answer a question using the buttons provided. The second player should then answer a question following this.	The game should display the value the user has selected and notify them that they are waiting for the second player. When the second player has answered the question both players are taken to the results screen, displaying the results.	PASS
Answer Easy Question Correctly	Answer an easy question correctly with 20 seconds left using the buttons provided.	The game should update the player’s score correctly by 70.	PASS
Answer Medium Question Correctly	Answer a medium question correctly with 40 seconds left using the buttons provided.	The game should update the player’s score correctly by 140.	PASS
Answer Hard Question Correctly	Answer a hard question correctly with 60 seconds left using the buttons provided.	The game should update the player’s score correctly by 210.	PASS
Answer Question Incorrectly	Answer a question incorrectly.	The game should not update the player’s score.	PASS
Answer Question Timeout	Do not answer the question at all, allowing the game to time out.	The game should not update the player’s score.	PASS

Test Name	Test Description	Expected Outcome	Actual Outcome
Prisoner's Dilemma Component	Simulate the actions: 1. Both players betray. 2. Both players cooperate. 3. Player1 betrays, Player 2 cooperates.	The game should update the player's scores by: 1. 0 2. 150 3. Player1 by 250, Player2 by 50.	PASS
Sealed Bidding Auction	Simulate the following actions on an item value of 100. 1. Player1 bids 50, Player2 bids 60 2. Player1 bids 50, Player2 bids 40 3. Player1 bids 5, Player2 bids 5. (Player1 bidding first) 4. Player1 bids 101, Player2 bids 50.	The game should update the player's scores by: 1. Player1 by 0, Player2 by 40. 2. Player1 by 50, Player2 by 0. 3. Player1 by 95, Player2 by 0. 4. Player1 by -1, Player2 by 0.	PASS
Trust Game	Simulate the following actions on an overall value of 200 and multiplier 2. 1. Player1 gives 50, Player2 returns 100. 2. Player1 gives 50, Player2 returns 0. 3. Player1 gives 0, Player2 returns 0.	The game should update the player's scores by: 1. Player1 by 150, Player2 by 100. 2. Player1 by 50, Player2 by 100. 3. Player1 by 100, Player2 by 0.	PASS
Ultimatum Game	Simulate the following actions on an overall value of 100. 1. Player1 gives 50. 2. Player1 gives 100. 3 Player1 gives 0.	The game should update the player's scores by: 1. Player1 by 50, Player2 by 50. 2. Player1 by 0, Player2 by 100. 3. Player1 by 100, Player2 by 0.	PASS
Complete Game	Complete through the main game loop of 15 minutes.	The game should end and present both players with the end game screen with their results.	PASS
Statistics	Play through the game and check the statistics	The statistics displayed by the game should match the outcomes of each minigame and question.	PASS
Feedback	Player through the game and view the question feedback.	The game should present the feedback of questions answered by the player.	PASS
Verdict	Play through the game and view the verdict from the game.	The game should present the correct achievements and ranks according to the relative boundaries.	PASS

9.9 Appendix I - Aggregated Answers to Questions

	Answers made per Question																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Answer 0	10	8	0	0	0	2	7	5	3	2	10	2	1	0	3	4	14	4	5	1	0
Answer 1	12	12	1	4	2	10	9	9	4	3	8	12	4	1	5	4	0	4	12	1	8
Answer 2	0	0	3	10	10	2	2		4	3		1	13	2	4	6	0	20	6	6	6
Answer 3	0	0	7	2	3	0	0		5	14		4	0	4	11	1	12			7	1
Answer 4	0	0	4	0	0	0			0	0		0	1	14		14	0			11	6
Answer 5	0	0	0	3	7	0			0	0		0									

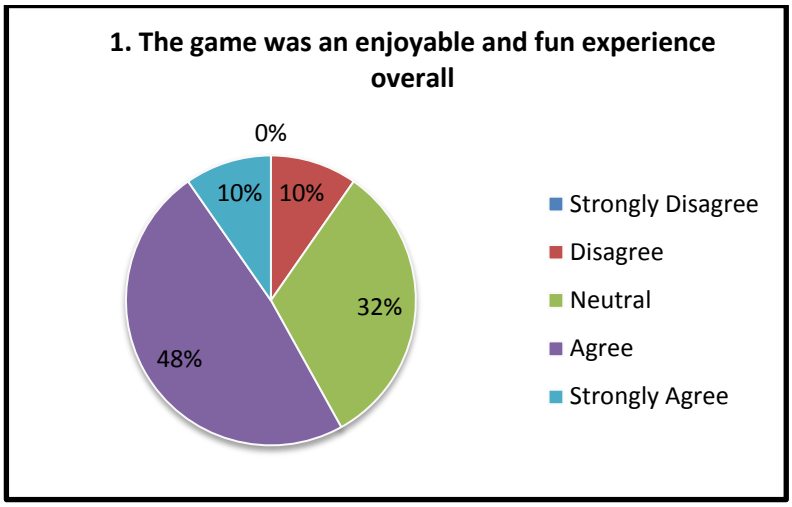
The rows represent the index of the different answers available for the multiple choice question. The columns represent the number of players that chose that particular answer for each question in the game. This table shows 21 questions that were fully tested and accepted by users. The green background represents the answer that is correct, all other answers are incorrect. More detailed information can be found in the Auxiliary Material attached with this report.

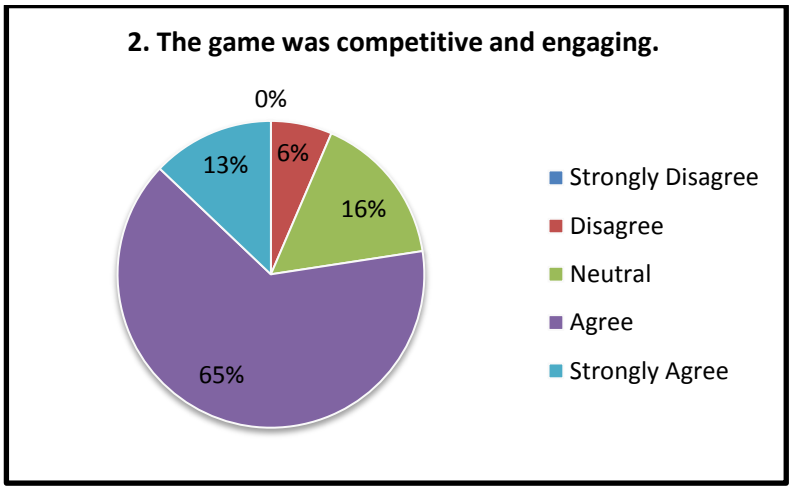
9.10 Appendix J – Game End Feedback Questionnaire

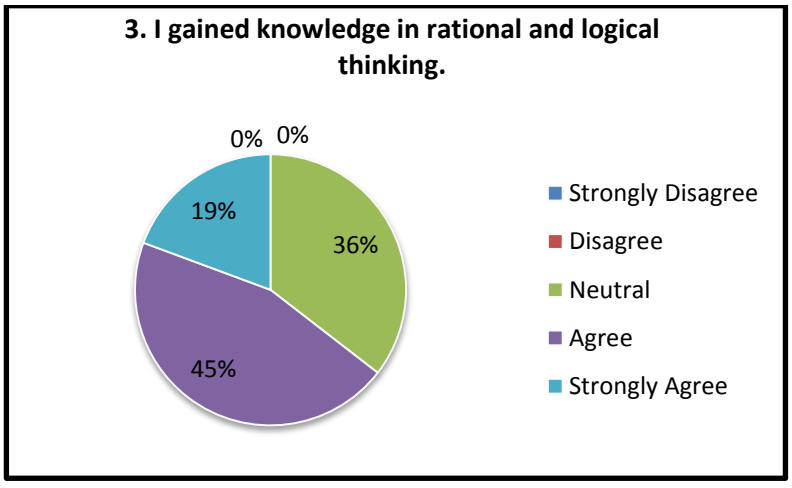
Page 1:

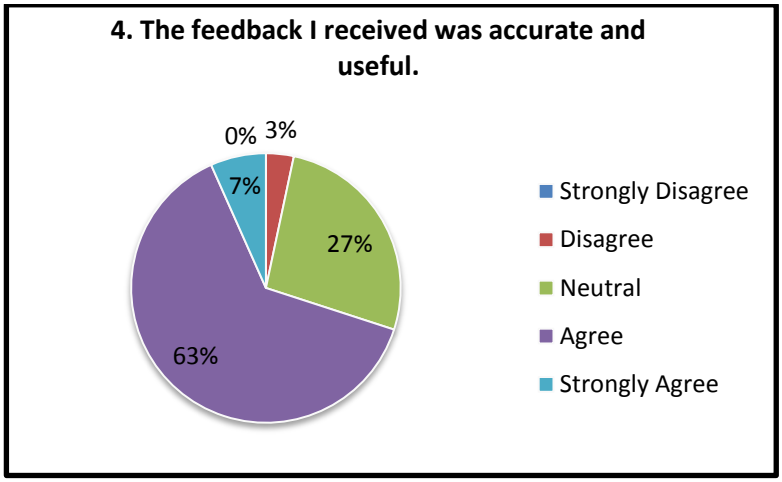
Page 2:

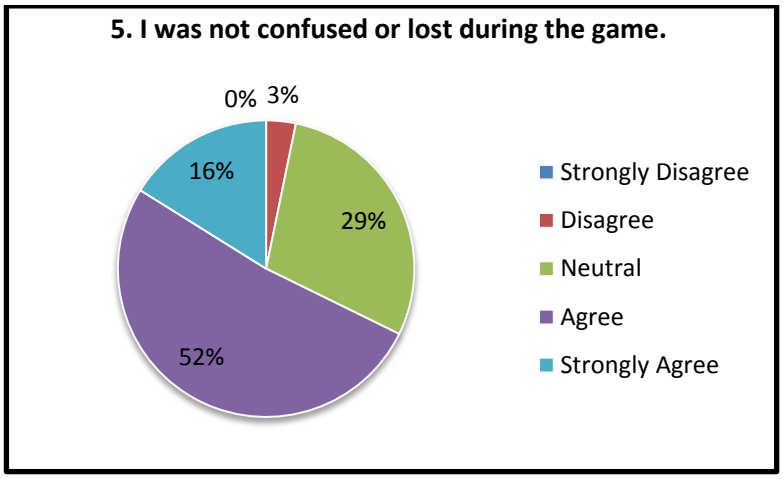
9.11 Appendix K - Game End Feedback Questionnaire Results

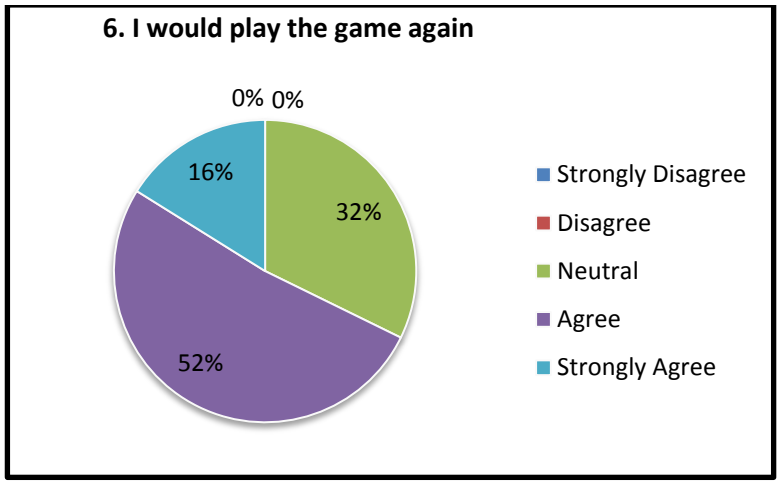







4. The feedback I received was accurate and useful.
0%
3%
7%
27%
63%
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


5. I was not confused or lost during the game.
0%
3%
16%
29%
52%
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


6. I would play the game again
0%
0%
16%
32%
52%
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree