

UNIVERSITY OF HERTFORDSHIRE

Faculty of Science, Technology and the Creative Arts

Modular BSc Honours in Computer Science

6COM0282 – Computer Science Project

Final Report

April 2012

Multiplayer Battleships

S F Fernandes

Supervised by: Paul Phillips

Abstract

The aim of this report is to discuss the development process of producing a multiplayer game of battleships. It'll go into detail about each of the stages from producing a feasibility study, to the production of the UML documents and finally the testing and evaluation.

There will be a brief discussion on the motivations behind gaming and how too much of it can cause to harmful addiction habits. It will talk about the statistics behind the gaming industry and how they may affect the game development industry.

This report identifies the achievements and downfalls of the project. There were several tasks that had to be carried out, most of which were successful with the exception of the leaderboard.

Finally, the report concludes the report with a discussion on the findings of the project and the learning experience undertaken. The lessons learnt from the start to finish will be outlined, ranging from the thoughts about the Java language to the use of the other tools and the approaches taken to the problem.

Acknowledgements

I would like to give big thanks to my project supervisor, Paul Phillips who has been very helpful throughout the entire project; providing valuable advice and going that extra mile to ensure all of his students have great resources available to them.

I would also like to thank all of my friends who provided inspiration and help whenever I needed it.

Table of contents

1 Introduction.....	5
1.1 History of the game.....	5
1.2 The game play rules.....	5
1.3 Variation of the rules	6
1.4 Variation 1: Ship sizes	6
1.5 Variation 2: Informing the Result of an Attack.....	7
1.6 Potential Variations:	7
1.7 Introduction to the Project Work	7
2 .1 Why play video games?	8
2.1.1 Introduction	8
2.1.2 Motivations behind gaming	8
2.1.3 Taking it too far	9
2.1.4 Conclusion.....	10
3 Requirements Specification	11
3.1 Functional Requirements	11
3.2 Non-Functional Requirements.....	11
4 Feasibility Study.....	12
4.1 Background	12
4.2 Issues to Be Addressed	12
4.2.1 What Tools Will I Need?.....	12
4.2.2 What Language Will I Use?	12
4.2.3 What Hardware Will I Require?	12
4.2.5 How Long Will I Need?	13
4.3 Recommendations.....	13
4.3.1 Programming Language	13
4.3.2 Integrated Development Environment (IDE).....	13
4.3.3 Graphics Manipulation Software	13
4.3.4 Hardware Recommendation	14
5 Artefact Design	15
5.1 Design Process.....	15
5.1.1 Use case Diagram.....	15

5.1.2 Sequence Diagram	15
5.1.3 Entity Relationship Diagram	16
5.1.4 Data Flow Diagram	16
5.1.5 Class Diagram	16
5.1.6 State Diagram	17
5.2 Design Patterns	17
5.2.1 Observer Pattern	18
5.2.2 Singleton Pattern	18
5.3 Design Artefacts	19
6 The Implementation	19
6.1 Implementation Process	19
6.1.1 Main Menu	20
6.1.2 Entry GUI	20
6.1.3 Leaderboard	22
6.1.4 Ship Setup	23
6.1.5 Single Player	23
6.1.6 Multiplayer	25
6.1.7 Game Settings	27
6.1.8 Server Browser	28
6.2 Source / Version Control	29
6.3 Security	30
7 Testing	30
7.1 The Purpose of Testing	30
7.2 Test Plan and Results	31
7.3 Test Result Findings	33
8 Project Evaluation	33
8.1 Design Evaluation	33
8.2 Artefact Evaluation	34
8.3 Testing Evaluation	35
9 Conclusions	35
9.1 Technical Conclusion	35
9.2 Project Conclusion	35

1 Introduction

Battleships is a popular game, where two players position their ships on a grid and then try to determine where the opposition has placed their ships. Each player takes it in turns to select positions on the grid in an effort to hit an enemy ship. The winner of the game is the first player to successfully hit and sink every enemy ship.

1.1 History of the game

Battleships was first introduced to the world predating world war one. In 1931, the Milton Bradley Company published the first official pen and pad game known as "Broadside, the Game of Naval Strategy". It would later on become a full-fledged board game in 1967. By 2012, there have been multiple variations of this game both electronically and as a board game for several different platforms including the PC, Nintendo DS and even the PlayStation 3. There is even a movie adaptation of the game being released in May, 2012.

	A	B	C	D	E	F	G	H	I	L
1										
2										
3										
4			X							
5						X	X			
6		X						X		X
7				X						X
8	X	X						X		
9										
10										

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

Figure 1 - An example pair of grids.
Left grid shows the player's grid, the right shows the opponents.
Source: The Wikimedia foundation.

For all intent and purposes, the final deliverable should be as close to the original version of the game as possible. Next we will look at the standard rules of the original game and to what length the game play rules will be altered in the product deliverable.

1.2 The game play rules

The most common rules of the game (also known as the Milton Bradley rules) will be used. They are as follows: At the start of the game, each player has an opportunity to place a series of 5 ships on a 2D grid, normally 10x10. When placing a ship, they must not overlap with another ship and they must remain within the range of the visible playing grid. Ships also may not be placed diagonally - only horizontally or vertically. Traditionally, these ships are of a fixed length. These sizes are:

Figure 2 - Traditional ship names and sizes

Ship name	Size (amount of tiles they take up)
Aircraft carrier	5
Battleship	4
Submarine	3
Destroyer	3
Patrol boat	2

Who makes the first move can be decided by the players themselves though using a random selector such as a coin toss is fairly common practice.

During your turn, a player must choose a coordinate inside of the opponent's grid. They may not choose more than one coordinate at a time and they may not select a coordinate that they have previously attacked.

If a player attacks a coordinate where an enemy ship has been positioned, the opponent must notify the attacker that the attack was a hit. Otherwise, they may simply indicate the attack failed and missed all ships.

Once all of a ship's positions have been hit, then an indication that ship has been destroyed must be presented to the attacked. Normally, a message such as "Ship Sunk" will suffice. Each player takes it in turns to attack an enemy co-ordinate and record the results until every single ship has been sunk. The first player to successfully sink every opponent ship wins the game.

A player is not allowed to lie about the results of an attack. If a ship is hit, then the truthful response must be provided to the attacker.

1.3 Variation of the rules

The final deliverable will use a slightly modified version of the Milton Bradley rules.

1.4 Variation 1: Ship sizes

The ship sizes will be slightly altered to the following:

Figure 3 – Proposed ship sizes

Ship Name	Ship size
Battleship	6
Cruiser	5
Destroyer	4
Patrol Boat	3
Scout	2

1.5 Variation 2: Informing the Result of an Attack

Like Milton's rules, players will still need to inform the attacker if a ship has been hit but they will not be required to inform the attacker if the ship has been sunk or not.

1.6 Potential Variations:

The grid size can be customized by the players to one of their own satisfaction. Usability factors will need to be taken into consideration when hard coding limits however.

Depending on the results of the feasibility study, players may have the ability to spoof the results of an attack to throw off the enemy. This will only be allowed a fixed number of times, probably depending on the grid size.

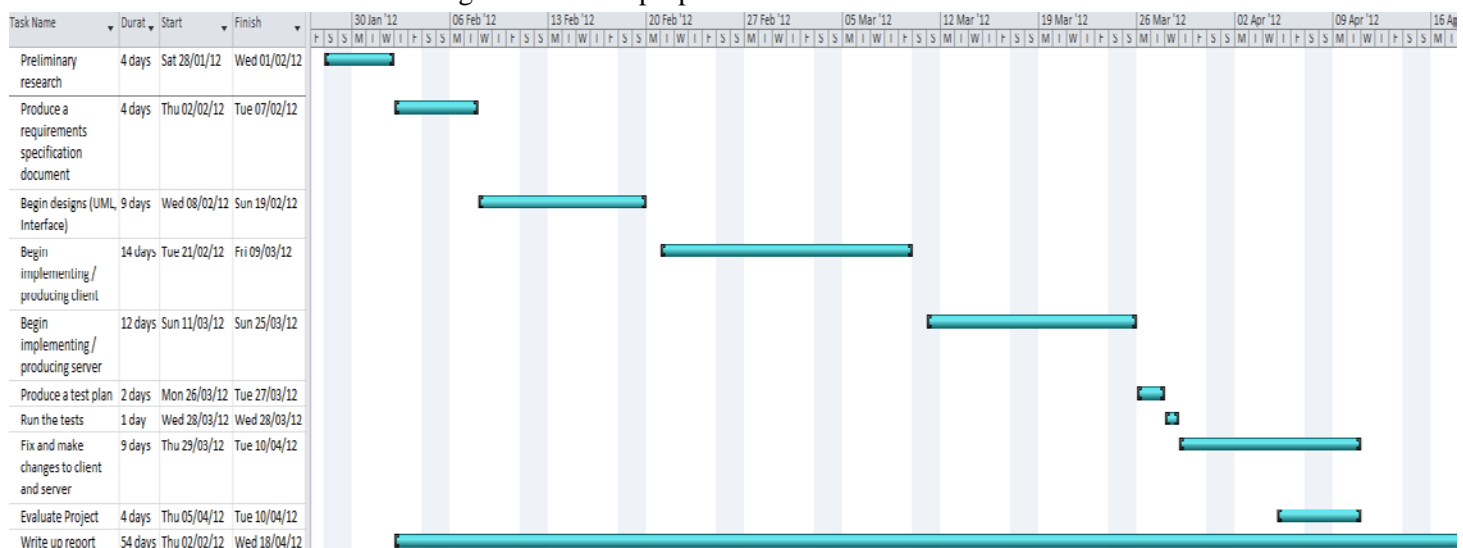
1.7 Introduction to the Project Work

The purpose of this project is to plan, implement, test and evaluate a digital version of Battleships that can be played over a network. Throughout a project cycle, there are many decisions that will need to be made, some of which will be difficult to make.

What is important is to be able to justify why those decisions were made and what the other options could have been. Such as, which technologies to use, what problems there might be in using those technologies and whether or not a work around solution is available.

The Gantt chart below shows the initially proposed schedule for the project. It specifically details which activities would take place on which days, with their estimated start and end times.

Figure 4 – Initial proposed work schedule



2 Literature Review

2.1 Why play video games?

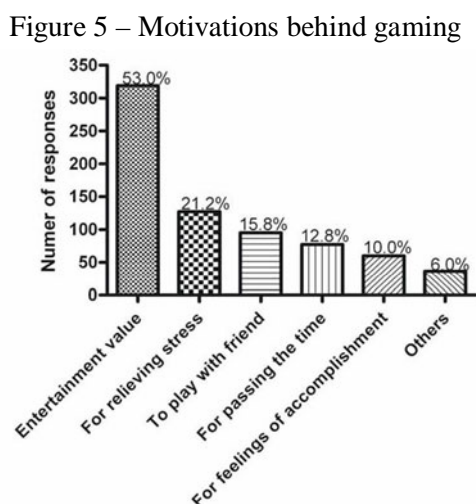
2.1.1 Introduction

Video, or computer games have become a massive industry since their first inception dating way back into the late 1940's. According to popular games coverage media site Joystiq, in 2011 the games industry reached an enormous \$74 billion in value and could increase to \$115 billion by the year 2015. It is pretty evident that there's a lot of money involved in the industry. Even during recessive and economic downtime, sales of video games still continue to surge and there is an increasing concern over the issue of game piracy. It is estimated that by the age of 21, an average American will have logged approximately 10,000 hours of gaming time. That is well over a year of time spent tackling objectives in virtual worlds that really amounts to nothing in reality.

Games are generally considered to be a luxury and their cost definitely reflect that, with new titles being released in the UK for an average of £39.99 per game on a console and only slightly cheaper on the PC. Games are products that people want, not products that people need, but considering the financial figures, the demand for this product is definitely there. This ultimately begs the question; if games are so expensive and they are not necessary for living, why do people play them? What exactly is it about games that drive people to shell out large amounts of cash for an entertainment product? Why buy a game over several movies?

2.1.2 Motivations behind gaming

Unfortunately, there have not been a lot of studies conducted and those that have been are difficult to find. From what I have managed to gather though, putting it bluntly, no-one really seems to have a definitive answer. Most initial reactions seem to be that games are simply used as a pastime, something to do when you're 'bored' or 'have nothing else to do'. But what is it about games that make them a preferred pass time option over other activities? Figure 5 shows the results of a study conducted by the authors of *CyberPsychology & Behaviour* in 2007. It shows what a group of 7069 gamers said that their motivations behind gaming were.



Source: *CyberPsychology & Behavior*, Apr2007, Vol. 10 Issue 2, p278-285

According to Tom Francis, game developer and author at PC gaming magazine, there are several factors that can contribute to whether a game will be appealing to an individual. A few of these are:

- **The challenge** - What exactly is the game asking you to do? Do you enjoy overcoming similar problems in the real world?
- **The feel** - How does the game feel to the player? Are the sounds, graphics and overall presentation of the game comfortable and can the user relate to the experience?
- **Freedom** - How does the game respond to your actions and command? Is the user in control of the game's direction?
- **Place** - Where does the game take place? Is it set in a fictional environment or is it based on true events? These questions could also apply to the game mechanics.
- **Fantasy** - Does the game let you do things only deemed possible in your imagination? Can you perform actions that you wish you could do in reality but can only really achieve in the virtual world?

Games are also deemed to have benefits to a person's health. An unknown author on TastyHuman.com cites several benefits of playing games as according to studies made. They claim that playing games improves a person's empathy, cognitive health, mood and in some cases it can even act as pain relief.

“Another study, by Sony Online Entertainment and Yahoo! who had a huge parent survey in June 2008 and found out that 70% of the parents surveyed have seen their children’s problem-solving skills improve since they started playing video games.”

“Scientists utilized positron emission tomography in order to show that levels of the neurotransmitter dopamine increased while playing video games (dopamine is believed to mediate several behaviours, one of which is the experience of pleasure).”

“One of the big things about many games is you’re interacting with other people in such a way that you have to actively think about what the other people are doing or thinking in order to either play against them or play them cooperatively. Either way you’ve got to be engaged in trying to think of how is this person learning and what’s this person going to be doing next.”

Source - TastyHuman, <http://www.tastyhuman.com/10-benefits-of-playing-video-games/>

2.1.3 Taking it too far

Despite all of these motivations, there are an alarming amount of reports reporting of increased levels of video game addiction. In some countries, for example Korea, internet use and gaming has become a part of everyday life, affecting not just young children and teenagers but affecting people of every age. The BBC reported in 2010 that a couple living in south Korea allowed their child to starve because they were too busy raising their virtual child.

“The pair fed their own premature baby just once a day in between 12-hour stretches at an internet cafe, the official Yonhap news agency reported. The couple had become obsessed with nurturing a virtual girl called Anima in the popular role-playing game Prius Online, police said on Friday.”

Source - BBC <http://news.bbc.co.uk/1/hi/8551122.stm>

It is pretty shocking to hear that the natural instincts to care for your child can be overridden by a desire to indulge in a virtual reality.

2.1.4 Conclusion

In my opinion, I believe that people play games not just as a pastime, but as an escape from reality and a chance to do all of these imaginative things that they cannot do in the real world. I feel that games offer players a chance to not only experience fantasy based worlds but more importantly, they offer players a chance to interact and feel a part of it.

I believe that games also provide an opportunity for people to experience these interactions socially, whether with people they know or people they have met online. A boom in cooperative and online multiplayer games provides valuable tools for people to satisfy their social desires.

I also feel that game developers have a moral and ethical obligation to ensure that gaming remains exactly what it is; a form of entertainment and social media. We must take care not to let the industry get out of hand and should do our best to educate people on the dangers of addiction and neglecting real world responsibilities.

How can we do this? I'm not sure. Perhaps encouraging players to take regular breaks or in the more extreme circumstances perhaps we should be prohibiting the amount of gaming hours an individual can spend. The fact of it is, is that with the extremely rapid rate of technological development, and our lives becoming more and more dependent on the internet and other virtual services, the likelihood of human beings allowing themselves to forget real world responsibilities is an ever growing concern.

All in all, the demand is there. Whatever the reason is, people are willing to pay good money for games and despite past horrors or disappointments, they will continue to invest their time, money and energy in tackling challenges and objectives that game developers throw their way.

3 Requirements Specification

3.1 Functional Requirements

These functional requirements are the key components of the game that must be implemented in some form or another. They are the primary features that should be included in the final deliverable.

- **Single player game** - A user should be able to start, play and finish a round of battleships in compliance with the standard battleship rules that the public have commonly come to understand. (Described variation of Milton Bradley rules)
- **Multiplayer game** - A user should be able to connect to another user's game session and seamlessly play the game over a network connection.
- **Host a multiplayer game** - A user should be able to start a game which allows one other player to connect and interact over a network connection.
- **View list of available multi-player games** - A user should be able to view a list of multiplayer games waiting for an opponent. They should be able to connect to a game without knowledge of the hosts IP address.
- **View leaderboard** - A user should be able to view a leaderboard of the most current up to date scores. Users scores should also be automatically submitted when a (multi player) game is finished.
- **Customization** - Users should be able to customize some of the games options. This could include the changing of your screen name to editing some of the graphical interfaces features. Alternate gameplay variables such as custom grid sizes should be included, too.

3.2 Non-Functional Requirements

- **Smooth and responsive** - The games commands, interfaces and operations should be quick smooth and responsive.
- **Usable:**
 - Learnability** - The game should be easy to learn and pick up straight from the start. The interface should be laid out in a way that users would expect.
 - Efficiency** - The game should be simple to navigate. Users should be able to perform tasks as quickly as possible.
 - Error Handling** - Users should be informed exactly of any errors they make and the necessary actions needed to take them.
 - Satisfaction** - The game all round pleasant to use. Do not make the user do something unnecessary and make each task as streamlined as possible
- **Rich media and interfaces** - The game should present itself in visually pleasing and make use of rich and entertaining media such as graphics, simple animations and even sound.

4 Feasibility Study

4.1 Background

The purpose of this feasibility study is to determine whether or not the proposed project is actually feasible. That is, can we make the game? Does the game need making? What will we need to make the game and how do we build the game? This section of the report will take a brief look at each of these issues and will look at potential solutions to each of the issues that are brought up.

The problem in this context is the creation of a Multiplayer version of the board game battleships in a digital format; i.e. a computer runnable program.

4.2 Issues to Be Addressed

There are several issues that will need to be addressed before development of the artefact can begin. Below, I will discuss these issues and address potential solutions for each.

4.2.1 What Tools Will I Need?

In order to create a game, I will need some kind of Integrated Development Environment (IDE). An IDE will be used to write, manage, compile and build the game during its development. What IDE's are available largely depends on what programming language is most suitable for the task at hand, so I will discuss the IDE once a conclusion on which programming language to use has been made.

For the creation of graphics, I will need some kind of image manipulation program powerful enough to allow me to make more than just minor changes to any images. The program should be able open and save images. It should allow me to manipulate the entire image any way I wish down to the pixel level.

4.2.2 What Language Will I Use?

In order to do any kind of programming, I need to determine which programming language that the game will be written in. The game needs to run fast. It must support networking capabilities and provide good support for reading and writing from files.

The program should be easy to run and not require any frustrating prerequisites before playing. The language should also be relatively easy to use. Having only 12 weeks, I do not have the time to learn a difficult programming language inside and out.

4.2.3 What Hardware Will I Require?

My hardware requirements are actually pretty low. I will require a machine capable of displaying some mostly static screens, perhaps some basic animations too. It will require a working network connection for the multiplayer component. The machine should also preferably run a modern version of Microsoft Windows.

For development, a machine capable of running an image manipulation program and an IDE is more than sufficient.

4.2.5 How Long Will I Need?

Based on previous experience developing software, I estimate I will require at least 8-10 full weeks to complete every component of the game to a respectable level. Depending on the amount of coursework I receive from other modules in my course, this estimate may increase.

4.3 Recommendations

4.3.1 Programming Language

There were a few main programming languages I could choose from. Java, C# and C++ were the most obvious choices. In the end, I believe Java to be the best programming language for the completion of the game. The reasons I have chosen Java are:

- **It's fast enough** - Java is still a pretty quick language. From my experience in using it in the past, I've never come across any problems in terms of performance when running any Java applications.
- **It's cross platform compatible** - Java is operating system (OS) independent. This means that as long as I don't use any windows specific libraries, the game will function properly on any OS that supports the Java Virtual Machine - which to my knowledge is practically all the popular ones.
- **I am comfortable and familiar with it** - I have been using Java for the past 4 years. I do not want to be spending the little time that I do have learning a brand new language. I feel I could get better results using a language that I know.

4.3.2 Integrated Development Environment (IDE)

For Java, there are two main IDE's that are used in industry. They are Netbeans and Eclipse. While there are many other Java IDE's, a good amount of them are not free and I do not have the budget to spend any money on software and as far as I'm aware, the university does not hold a license.

For this project, I will be making use of the free and open source IDE Netbeans 6.7 and version 7. Frankly there is only one reason I have chosen to use Netbeans instead of Eclipse and that is because I have spent the last year learning how to use it in my in one of my course modules.

4.3.3 Graphics Manipulation Software

There are three obvious choices for graphic manipulation software. Adobe Photoshop, GIMP and Paint.net. GIMP is a good free program that supports many of features that I required and Paint.net also offer similar functionality. They are not bad choices and if I was conducting my own business, I would seriously consider using either program as the main choice. However, for the sake of this project, I have decided to stick with using Adobe Photoshop CS5. Here are my reasons why:

- **Photoshop is available on all university machines** - As far as I can tell, the university computers do not have Paint.NET or GIMP installed by default despite being free. If I start work on an image, I expect to be able to continue in the same environment regardless of what machine I'm at.
- **Photoshop is the industry standard** - Photoshop can pretty much do anything you could want to do with an image. If there was a task that I need to do and haven't thought of it early on, chances are Photoshop will do what I need it to do.
- **Lots of experience** - I actually have a good amount of experience using Photoshop. I've used it since version 7, which was released several years ago and have had continued use with it throughout that time. Performing tasks with Photoshop would be much quicker and easier than using software that I'm unfamiliar with.

4.3.4 Hardware Recommendation

For the vast majority of the development, I will be using the computers that the University provides in its Learning Resource Centre and in its labs. The computers there are pretty fast and are equipped with all of the software that I will require to complete my work. Because the entire university is networked together, it will also make it easier to test my games networking capabilities when that time arises.

5 Artefact Design

In this section, I will discuss the process undertaken when designing the game. I will talk about the decisions that needed to be made, which decision was made and why. I will be showing the different UML artefacts as well as a brief explanation for each and what they represent.

5.1 Design Process

The first step in the design process was to decide which modelling and design techniques I would use. In order to determine which of the UML models to work with, I looked at the various different diagrams that exist and then broke down the advantages and disadvantages of using each. Next, I will list the most common models (that I am familiar with) and explain why I did or did not chose to use the model when producing my design.

5.1.1 Use case Diagram

The primary function of a use case diagram is to capture the requirements that a system, or piece of software should have. The main idea is to show each of the tasks or actions that an 'actor' (someone who interacts with the system) should be able to perform.

Advantages for producing a use case diagram:

- It is simple to produce and follow. It is designed so that anyone who reads it can instantly understand what each actor of the system is supposed to be able to do and under what conditions they can do it.
- Use case diagrams bring out the obvious system needs early on in the project, allowing the developer to know exactly what the finished product should be able to do.

And the disadvantages

- Writing the documentation that comes with the use case diagram can be long and tedious to write. In a project where only 6 weeks is available.
- The use case can only model requirements correctly if the requirements for the system are fully finalized and properly understood. Requirements are likely to change mid-development!

I decided to use a use case diagram for my project because it was the easiest model for modelling the system requirements for my game, and I have had a lot of experience producing the model throughout the duration of my university course. Knowing exactly what my system needs to be able to do provides a good basis for producing a test plan. If you know what the functionality is supposed to be implemented, you can test to see if it performs as expected.

5.1.2 Sequence Diagram

A sequence diagram is used to show the sequence of operations that a particular method or system function follows when being executed. It shows the order that each operation is called and which objects it needs to interact with to complete its task. It is particularly useful for solving problems that require complex logic. When deciding which models to use, the sequence diagram occurred to me as being a useful tool to show the interaction between a client and game browser server when registering that a new multiplayer game had been started.

Advantages for using a sequence diagram:

- It practically provides you with a step by step instruction set for performing an operation. If the diagram is good enough, it can literally tell you what code to write, what each method call should do and what they should return, if anything.

Disadvantages of sequence diagrams:

- Modelling every complex process in the system would take an enormous amount of time. It's just not feasible.

Due to its incredible use for solving operation logic, I decided to produce a couple of sequence diagrams to model the process for registering multiplayer game information to the server and back.

5.1.3 Entity Relationship Diagram

An entity relationship diagram (ERD) is used to model the structure of tables in a relational database. They show a databases entities, relationships, and attributes. They are useful for describing how the tables interact, and they help you to produce an efficient normalized database.

Advantages of an ERD:

- Provides a visual representation of a database's structure, its entities and relationships. An ERD is invaluable to produce a normalised working database.

Disadvantages of an ERD:

- ERDs only show the structure of a database. They do not provide modelling for the manipulation of data that the database might be holding.

Although my game made use of a database for the high scores table, the structure of the database was so incredibly simple that there would have been absolutely no use for an ERD. The database will most likely only have a single table in it so there's not really anything to model and it would have been a complete waste of time.

5.1.4 Data Flow Diagram

A dataflow diagram is used to show the flow of information throughout a system and which processes or methods manipulate that data and what it does with it. It also shows where data is stored and where it enters and leaves the system. It is generally an easy to read model and there is normally no technical expertise required when viewing and trying to understand the model. It's the kind of diagram you'd show your manager!

Advantages of a DFD

- Easy to produce and easy to understand. You could show one to anybody and they wouldn't have much trouble understanding.
- Provides a visual system wide view. DFD's also show the system boundaries - external sources that cannot be modified.

Disadvantages of DFDs

- Physical considerations are left out.
- Many levels of DFD's are required to cover the entire system scope.

I decided against using a DFD because I did not feel that modelling the entire system was necessary as there really aren't that many different functions in the game. You can only play a single player game, a multiplayer game and change some settings. Having a diagram to model this just seemed excessive and a little over the top. I did not find a DFD to be appropriate to produce for this project.

5.1.5 Class Diagram

A class diagram is used to show a visual representation of a program's classes, class relationships, class method and fields and more. The class diagram illustrates the structure of a program, rather than the flow of information or the order that a process may be executed in. They are an invaluable tool when designing an object oriented implemented piece of software, they help you to design extendable, cleaner and efficient code.

Advantages of using a class diagram:

- Leads to a robust program
- Forces the developer to think exactly about how the program is going to work. If it cannot be modelled, it probably can't be programmed.

Disadvantages of using a class diagram:

- The class diagram is always changing. There is no way of knowing what the initial class design will actually work in practice, so you tend to update and change the diagram as you go along.
- Large pieces of software may require extremely large and complicated class diagrams. Sometimes, it may just be better to produce multiple smaller diagrams modelling a specific part of the program rather than just bunch everything into one.

I decided to produce a class diagram, because I wanted to have a good idea of what the structure of the game would look like and to have a decent idea of what classes I'd need to have and to know what their roles and responsibilities throughout the system are.

5.1.6 State Diagram

A state diagram is used to show the different states that a piece of software can be in at any given time. It shows the starting point(s) and ending point(s) of a system. It is a behavioural diagram and it represents the consequences of any actions being made by an actor.

Advantages of a State Diagram:

- Shows each possible outcome for each action
- Easy to read and understand and it's not very time consuming to make
- Fantastic reference point for creating a test plan

Disadvantages of a State Diagram:

- Only useful if the systems functional requirements are set in stone.

I decided to produce a state diagram as I felt it would be a good way to model the interactions of the game's graphical user interface. More specifically, it would help me determine what button does what under what condition. Being a relatively small game, the transitions a user can make would be down to a minimal level and therefore would not take too long to produce.

5.2 Design Patterns

When producing my class diagram, I tried to think of ways to make the code extendible and reusable. One of the ways that I have learnt to do this during my studies at university is to program to existing design patterns. According to an article on Wikipedia, a software design pattern is "a description or template for how to solve a problem that can be used in many different situations".

Source: http://en.wikipedia.org/wiki/Software_design_pattern

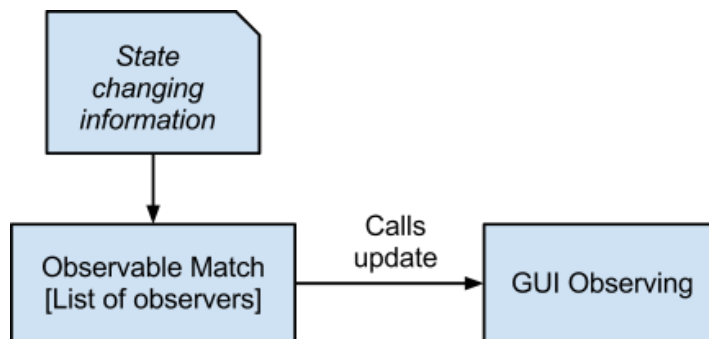
Below, I will list a couple of the design patterns that I used the most. I will give a brief explanation of what each is supposed to do and where and why I used them.

5.2.1 Observer Pattern

The observer pattern generally consists of an observable object which contains a list of other objects that are interested in any state changes of the observable object. When the state of the observable object is changed, the observable will call a method on each of the observer's objects. Java does contain built in support for an observer/observable implementation and I made use of these classes in my game.

In the game, I primarily make use of the observer pattern to notify the match GUI that a piece of information has been received. This information could be anything from a message received to a move being made by the opponent. Figure 6 shows a representation of this behaviour.

Figure 6 – Observer, observable updates



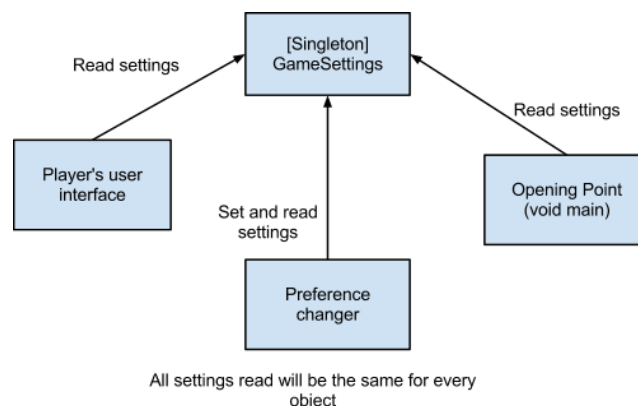
The observer pattern provides a good way of solving the object notification problem that would have arisen without it. How else would a GUI have known that the status of a match had changed without asking it every second or so?

5.2.2 Singleton Pattern

The singleton pattern is a design pattern that ensures that there is only ever one instance of an object and that is it accessible throughout the entire application. It is useful for providing some type of global state and in my game; it was used to handle a difficult sound threading problem. I also used the singleton pattern to load and provide access to any external resources to the program, such as images, game settings and sounds.

Figure 7 shows a brief view of how the singleton pattern was beneficial in handling game settings.

Figure 7 – Benefits of the singleton pattern



5.3 Design Artefacts

The completion of the design left me with some design artefacts that I could use as a reference point while coding the games implementation. The final design deliverables were:

- A set of class diagrams
- A use case diagram with an accompanying list of use case descriptions
- A sequence diagram showing the sequence of operations for registering a new multiplayer game
- A state diagram showing all the possible states that the game can be in.

Please see the attached appendix for a copy of all of these design artefacts.

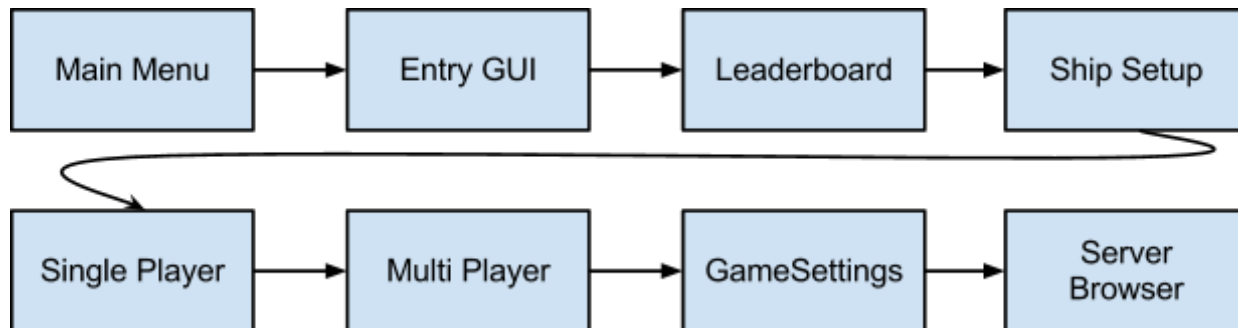
6 The Implementation

In this section, I will discuss the implementation process I took and the steps that were taken to complete the final artefact. I will discuss what order each of the components I worked on were finished and why. I will talk about some of the technical challenges I encountered and how I overcame them.

6.1 Implementation Process

Once I had all of the design documents completed, it was time to make a start on coding. I decided to start off with the easier tasks such as the menus and graphic creation. Figure 8 shows a diagram of the implementation process that I used (which components I did in which order)

Figure 8 – Implementation process



Next I will discuss each of these main components individually. I will talk about the issues that arose and how I overcame them.

6.1.1 Main Menu

The main menu was the very first component of the game that I worked on. I decided to work on the menu because I felt it was going to be the quickest and easiest part to get done first. The first step was to get a size and layout that I liked. In order to have a custom background image, I needed to extend a JPanel and overwrite its `paintComponent()` method. This gave me access to the JPanel's Graphics object allowing me to customize the way that the panel was rendered on the screen. Originally, I was not too sure on how exactly the Graphics object worked. It took a couple of hours of reading tutorials to understand exactly what the methods do and how to use them relative to the parent JFrame.

Another issue I had was with the GUI layout manager. Netbeans offers a drag and drop interface for adding common objects such as buttons and labels. However, when I was trying to layout all of the buttons and labels, I found that occasionally the placing of a new object caused a bunch of other objects to change their positions seemingly randomly. Very irritating to say the least. Eventually I worked out that it was due to the way Netbeans drag and drop manager likes to 'snap' things in line with other objects. A bit of playing around and I finally managed to get it to look the way I wanted.

The biggest issue I had developing the main menu was implementing the music and the music toggle. The music player makes use of an open source third party library called JavaLayer. Implementing the music was surprisingly difficult because the library came with little to no documentation on how to use the classes and it soon became a guessing game on how the implementation of these methods were supposed to work. I managed to get the music toggle to work by making use of a thread that implemented the singleton design pattern. This meant that only one instance of the music thread could ever exist, allowing better manipulation of the thread and its methods.

To develop the actual background image for the main menu, I found a few good looking high quality images on Google's image search and used Adobe Photoshop 5 to glue together a brand new image. A couple of revisions later, and the background image that would be used for my game had been made. Figure 9 shows what the final version of the main menu looks like.

Figure 9 – The main menu screen



6.1.2 Entry GUI

The entry GUI was the second component of the game that I worked on. It's only real function was to allow a cosmetically pleasing way for the player to enter the name that they would be referred to everywhere else in the program, more specifically on the leaderboards and in multiplayer matches. Like the main menu, the Entry GUI

composes of a JFrame and a JPanel, which has had its paintComponent() method overridden is an extended class. The background image was also made in Photoshop, although this time the graphics was made entirely by me.

I did make use of some free non-commercial fonts for the Battleships logo and other lettering. Because of the lessons I had learnt from constructing the main menu, building the entry GUI posed no real problems and worked pretty much first time.

You may think that because the entry GUI is the first thing that appears when the program starts, that perhaps constructing the entry GUI first would have been a more logical approach. The reason that the entry GUI was not created first is because originally, I had not intended for there to be an entry GUI in the first place. In the early versions of the game, there was simply an option pane that appeared when the program launched, which prompted the user for their name. Adding the entry GUI was a decision I had made after the main menu was constructed for the sole reason of a better user experience.

Figure 10 shows what the final version of the entry GUI looks like.

Figure 10 – The first prompt shown to the user



As the name suggests, the entry GUI is the only ever shown one time at the start of the game. Should the player every wish to change their nickname, an option was added to the main menu which shows an input prompt. I felt it would have been a bit overboard to display this entry GUI every time the player felt like changing their name.

I would also like to note that by default, the input box does not allow users to hit the return key on their keyboards to trigger an action. This all had to be hand coded manually and was implemented for the sake of accessibility and ease of use.

6.1.3 Leaderboard

Next up, we have the leaderboard. I decided to implement the leaderboard before starting the actual playable game because I thought it would be useful to have the database construct ready for score submissions when the time came to coding the end of a match. The leaderboard was created in the same way as ever other GUI in the game, so there was no real issues in making it look right.

The leaderboard has been hooked up to an external MySQL database. The database layout is extremely simple, only one table with 3 columns. To establish a connection to the database, I had to make use of Oracle's MySQL database drivers which are publically available and completely free of charge.

Due to prior experience working with MySQL databases, getting the connection set up and reading from the database proved to be a simple enough task. With the connection made and data coming in from the database, it was a simple matter of getting this information displayed to the player correctly. Figure 11 shows what the leaderboard looks like, displaying some test data.

Figure 11 – Leaderboard displaying information from the database



Name	Score	Date
ChickenMan	45345	2012-02-15 13:27:24.0
Max	234234	2012-02-15 13:27:08.0
Test1	345345	2012-02-14 14:54:56.0

Unfortunately, due to time constraints I was not able to get score submissions implemented. I decided to keep the leaderboard in the game to demonstrate my ability to read from a database over a network and display its results. Perhaps sometime in the future, I will find the time to add a score submission system and then the leaderboard will finally have some real use.

6.1.4 Ship Setup

After the leaderboard was complete, I felt it was time to commence of the main gameplay component. Before players could engage in a match, they would first need to lay down their ships on the grid. The first step was to get interface to look good. This was a much bigger challenge than the other GUI's because this time, there was a large grid that could dynamically change size to account for.

My original plan was to have a set of ships that the player could drag and drop onto the grid. I did some research on the web into how exactly it was that I could achieve this, but after several hours of researching I still could not come up with an appropriate plan to have this implemented. Time was being wasted, so I decided that to compromise, I would make the user click on two points; the start point and the end point. The system would then have to figure out if the ship placement was a legal option and would inform the player if it was not.

Soon after starting, I had a system working where the player could select the positions and the system would place it correctly. Players would have to lay the ships down in a specific order. First the six size, then the 5th, 4th, 3rd and finally the second. It became apparent though that it was easy to forget where the first position that you had clicked was, even though there was a textual representation of the points you had clicked. To remedy this, I decide to implement a highlighting feature that would essentially show you all of the available points that you could select, based on the first position that you selected. Figure 12 shows a demonstration of this feature.

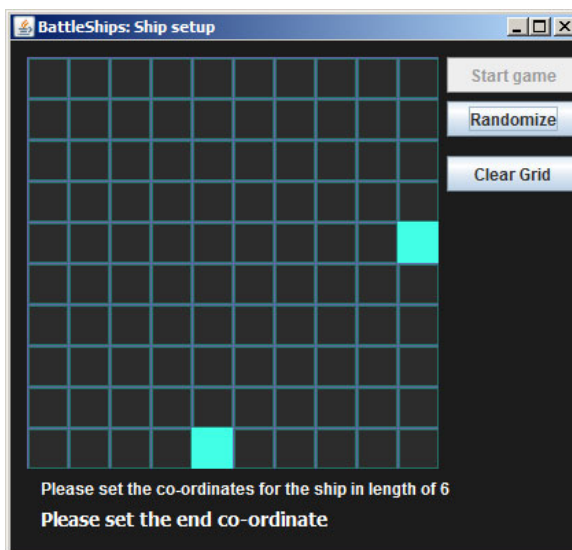


Figure 12 - Placement markers

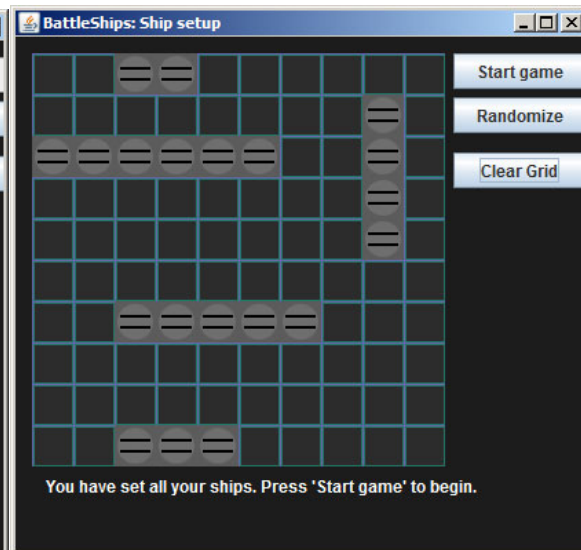


Figure 13 - Ships placed

A couple more features were later introduced. An option to clear the grid was added. Like the name suggests, this option completely removes all the ships that are currently on the grid and lets the player restart the ship placement from the beginning.

Once the single player component of the game was finished, I used the AI's ship positioning function to implement a randomize feature on the ship setup screen too. This button can be clicked as many times as the user wants; each time clearing the screen and randomly placing ships within the grid.


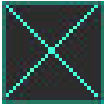


6.1.5 Single Player

The single player component was one of the hardest components of the entire project to implement. It had to work allow two dynamically sized grids to be placed and also needed to take into account other objects on the screen such as the chat window and message box.

The first step was to create the interface that the player would see when playing. I placed the two grids, side by side. The left grid is there to indicate the player's own grid and would show the position and status of each of the ships that were positions back in the ship setup menu. The right grid would indicate the current status of the opponent's grid (in this case the AI) and would only reveal parts of the grid as the user selected tiles to attack.

Once the GUI was displayed properly and had proper references to the player's grid and the opponent's grid, it then became a matter of registering the player's moves and displaying the result back to them. I decided the best way to do this was with a variation of the icons that were used to represent the default tiles. The table below indicates each tile in the game and what they represent (colours are subject to the players preferences)

Figure 14 – Tiles used in the game

Empty / Unknown	Missed Attack	Ship Tile	Ship Hit
			

In order to determine if a player had won, every time a ship was hit, the total amount of remaining ships would be subtracted by 1 until eventually; one of the lists of ships would hit 0. The first player to hit a ship tile count of 0 was the loser of the match.

The artificial intelligence (AI) for single player is actually extremely simple. Every time the computer takes a turn, it selects a point from a list of currently remaining positions and attacks the tile. It does not try to find ships based on whether the last attack was a hit or not. I did have the intention of making the AI smarter, but I was getting very conscious of time and decided it was better to invest my time in having a complete working game rather than a smart AI. After all, my project was about making a game of Battleships, not developing a smart AI system.

Developing single player did allow me to make a single improvement elsewhere in the game though. When playing against the AI, I needed to create a method that would randomly place ships on a grid. Using this method, I was able to create the 'Randomize' feature on the ship setup component much more easily.

Originally, I had planned to have the result of each move display in the message window. During gameplay testing though, it had occurred to me that it was actually more of an irritation than a helpful feature. I decided to keep the message area in though as I knew that the multiplayer component of the game would make use of it. The area appears in single player but it serves no real purpose at the moment. I could however, make it an option to enable logging in the future.

Figure 15 shows an annotated screenshot of the final version of the single player match interface.

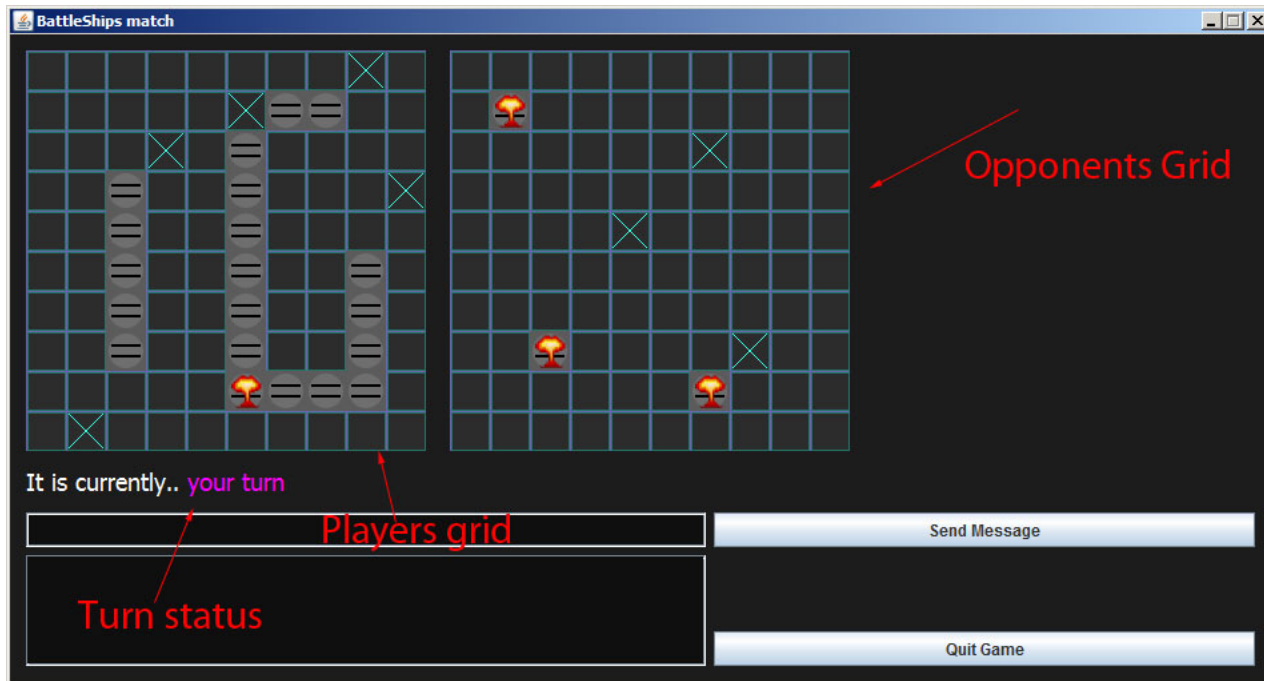


Figure 15 - The single player interface

Once single player was deemed complete, I decided it was a good idea to go back and refactor some of my code. The following is a list of the refactoring tasks undertaken before I felt it was time to move onto multiplayer.

- Refactored the tiles so they are drawn by Java and not by using a local image. This laid the foundation for coding the games custom preferences feature later on.
- Added the randomize feature to the Ship Setup GUI.

6.1.6 Multiplayer

The next component of the game to be completed was the Multiplayer component. Surprisingly, coding the game to work over a network was surprisingly simple. I did struggle a little with the host handling, but that was overcome pretty quickly.

Under the hood, the multiplayer component is pretty much an instance of a single player game. The only difference being the functionality of some of the buttons and where to get the opponents moves from. Instead of having the computer pick a position, the game would need to wait for a packet to be sent by the opponent. For this reason, the multiplayer match interface looks virtually identical to the single player window.

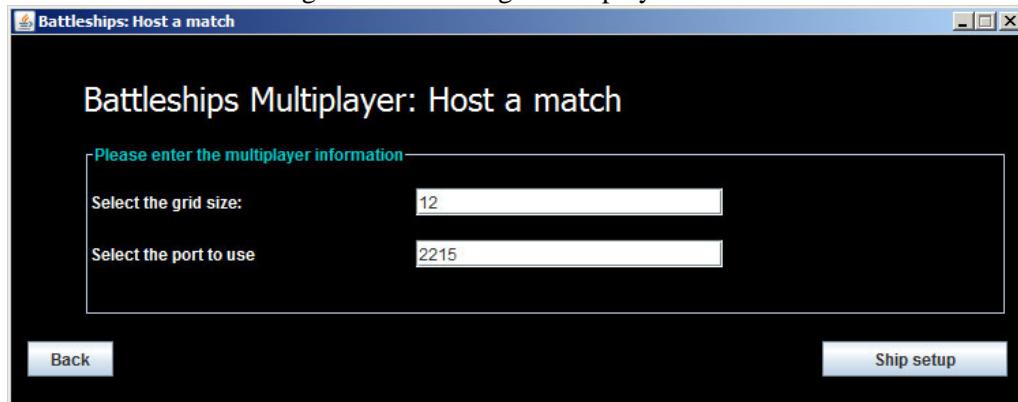
There were two main problems I needed to solve with multiplayer matches. The first was joining a game and the second was hosting a game. The problem was the fact that a multiplayer match was an extension of a single player match, and therefore some methods had to be completely rewritten and some methods had to be added. It also occurred to me mid-way through that I would need to add some empty methods in the single player class, just so the multi-player class could inherit the function calls. A bit of a headache, but the entire thing had been fixed and sorted in about a day.

In order to keep the player in the loop, I needed to provide feedback on the games current status. I therefore put the entire game server handling code in a separate thread and produced a very basic GUI designed to inform the

host of the current status of the server.

Figure 16 shows the information that the host needs to provide when creating a server.

Figure 16 – Hosting a multiplayer match



Figures 17 and 18 shows the messages the host sees during the two different stages of hosting a game.

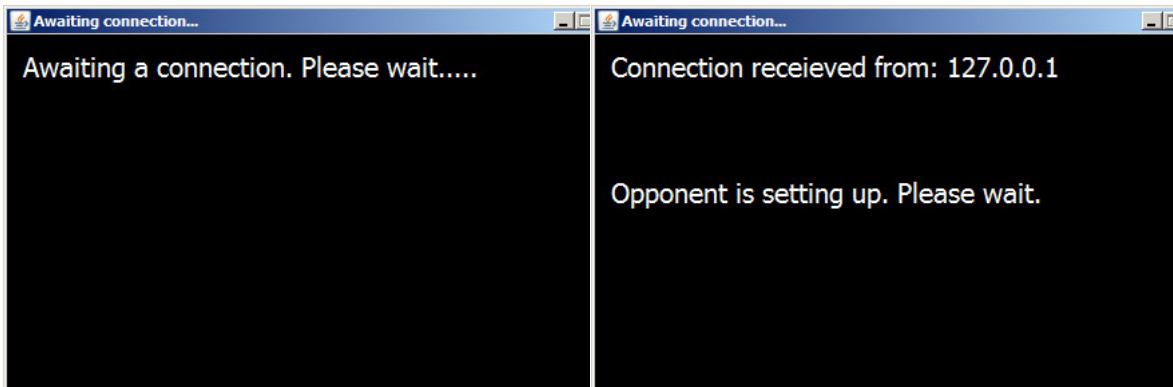
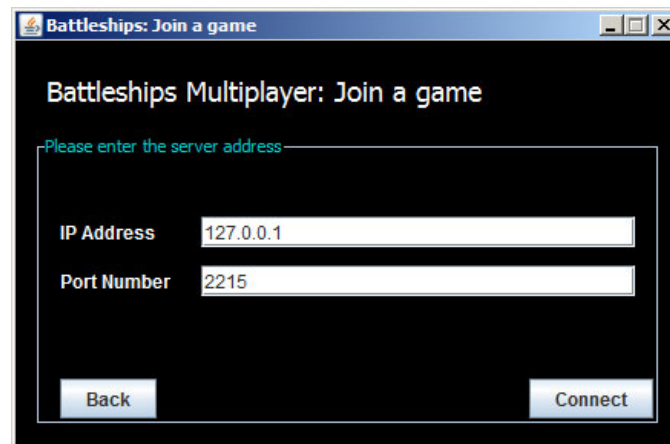


Figure 17 - Waiting for a connection

Figure 18 - Connection has been received

The second part I needed to handle was a player connecting to a server. There was one of two ways a user would be able to do this. Direct IP or via the server browser. However, during the development of the multiplayer component, the server browser had not yet been made - that came later. The player enters an IP and port of a known game host. If the connection can be made, the client receives information from the host and can then begin to set up their ships. Figure 19 shows the interface that a client is shown when they want to connect to a server.

Figure 19 - Joining a multiplayer game by IP



Lastly, multiplayer matches have the capability to transmit and receive player written messages to each other. This is where that empty message box from single player finally comes into play. Players can send and receive messages at any point during the match, regardless of whose turn it is. Figure 20 shows an asynchronous chat session going on between a client and the host of a match.

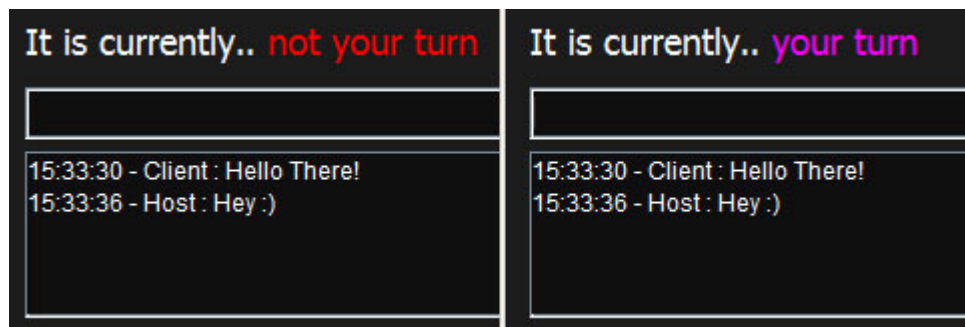


Figure 20 - Communication between two players

The multiplayer component was really the core aim of my project. The main objective was to develop a game of battleships that could be played over a network and this section proves the completion of that task. The only reason I completed the single player component first, was because in doing so, it would allow me to make use of a games foundation that could simply be extended to receive player's moves over the network and that's exactly what I have done.

6.1.7 Game Settings

Once multiplayer had been completed and most of its bugs were fixed, I decided to make the game a little more customizable. I wanted to give the player the option to changes the colours of the grids to anything they felt fit, and to also give them the option to change it at any time - even during a match. Thanks to my previous refactoring, which resulted in the removal of images being used and instead using Java's Graphics object, I was able to specify the colours of tiles and their borders.

To allow the user to do this, I made a very simple GUI which contained multiple colour pickers for each of the tiles aspects. Upon changing a colour, a sample grid would update itself showing the player in real time the effects of their changes.

Figure 21 - shows the colour picker in operation.

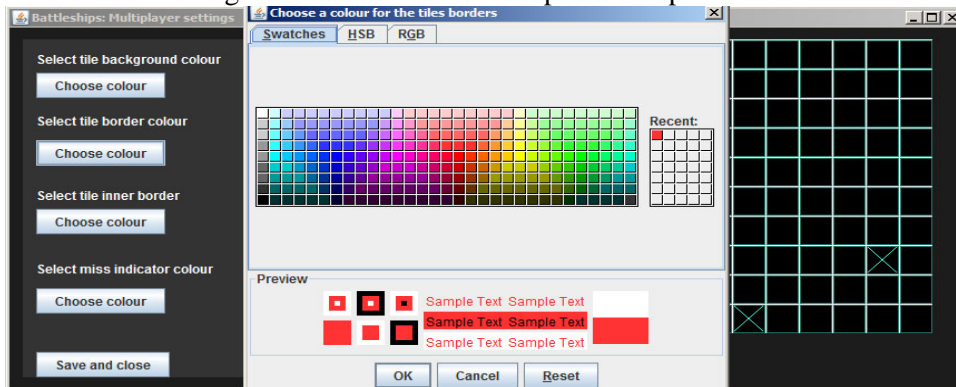
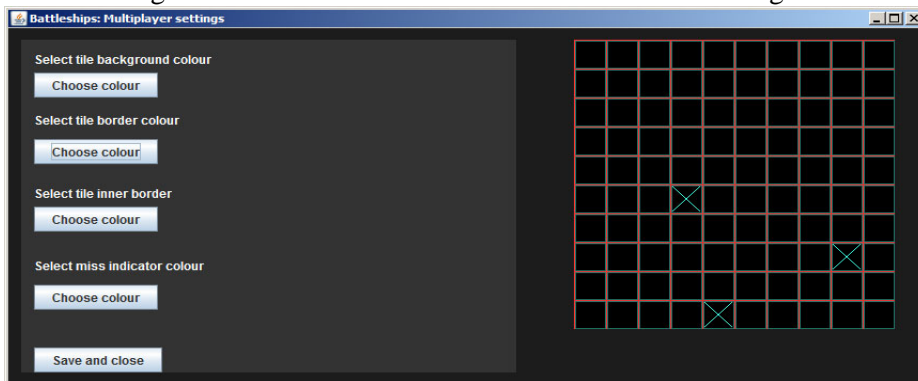


Figure 22 - shows the instant results of the colour change.



When the user selects the colour, the grid is updated to reflect the changes. This is true for all of the settings shown in the above figures. It's also worth noting that these settings are persistent. When the game first loads up, the game will check to see if a settings file exists and if it does, it will attempt to read the settings stored in the file. If no settings are found, the game will set default values. All of the code that handles the game settings (including the file reading) is declared in the GameSettings class statically. This means that it is available everywhere in the program, not just in the settings screen.

I did not really encounter many issues when handling the game settings component. The only issue I had was with the file reader. After several failed attempts, it turned out that the reason that the settings were not saving correctly was before I had forgotten to close the BufferedWriter. Amazing how missing one line of code can cause you hours of headache and frustration!

6.1.8 Server Browser

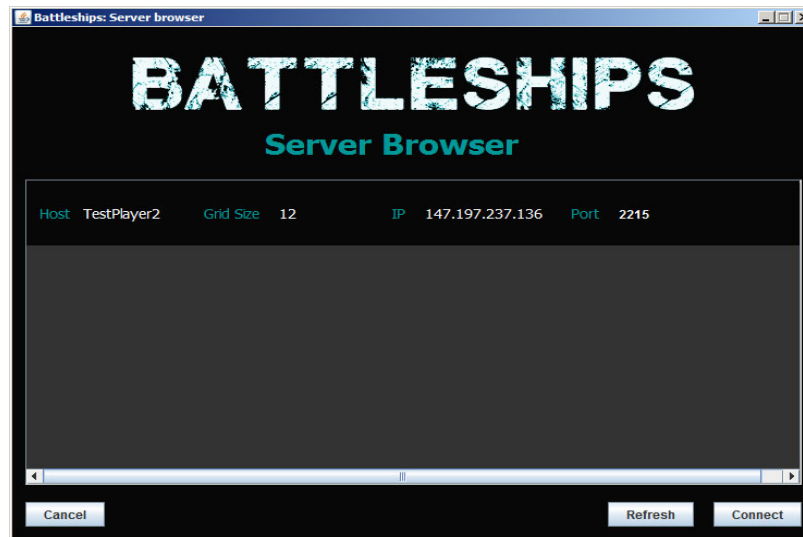
Finally, we come to the final component of the game - the server browser. The server browser was originally not going to make the final version of the game. However due to the quick completion of the networking/multiplayer component, I decided to give the server browser a go anyway. In order to get it working, I needed a server that would keep track of current available games and another GUI to display the list of games to any clients who may want to join.

Firstly, I tackled the server. I started it as a standalone project in Netbeans because I wanted an easy way to launch it on any machine without having to use a command prompt. The server was actually completed relatively quickly, although admittedly it is very simple. It doesn't really have much validation or error checking but the

point is; it works. To get the server to work correctly, I needed to go back to my multiplayer classes and edit some of the code, so that when a host is waiting for a connection, it sends the server the information on the game and when the game commences, it tells the server to remove it from its game list.

Once the server was working properly, I started work on the GUI that would display a list of games that the server was holding. The idea was that a player could simply select on a game from the list and clicks connect. No need for manual IP or port entering! Figure 23 shows what the final version of the server browser looks like.

Figure 23 – The server browser



Each selection shows the hosts name, the grid size, the IP and the port of the game. A player simply selects one from the list and clicks connect to join the game.

6.2 Source / Version Control

Source or version control is a software implementation designed to keep track of revisions made to a piece of software. It allows multiple programmers to work on the same piece of code while keeping track of every single change ever made by each programmer. This is extremely helpful for determining which piece of code broke the program and if needs be, it also allows you to revert to an older revision.

At the start of my project, I originally had no intention of making use of version control. Seeing as I was developing the project on my own, I didn't see the need to make use of it. It was only roughly a third through development did I decide to start using version control. I decided to make use of a free service called Assembla Workspaces. Assembla workspaces are a company offering free and paid for version control packages and in my particular case; I decided to use Subversion (SVN).

I am so glad I committed my project to SVN, because shortly after subscribing, my program started to behave erratically. It was throwing all kinds of nonsense errors and I have a strong feeling the source files had somehow become corrupted. Thankfully, I had a copy saved on my SVN account and was able to recover my entire code base with very minimal loss. Large industries use version control for a good reason and I experienced that reason first hand during development. I will be sure to use SVN in all of my future projects. Figure 24 shows what the final log for the SVN looks like.










Name	Date	Rev.	Commit message
 AI	Mon, Mar 12	3	[Simon Fernandes] Imported the game
 battleships	Wed, Mar 28	27	[Simon Fernandes] Fixed single playing quitting not working. Re-enabled the...
 Browser	Mon, Mar 26	23	[Simon Fernandes] Server browser now works
 Leaderboard	Mon, Mar 12	3	[Simon Fernandes] Imported the game
 Multiplayer	Wed, Mar 28	27	[Simon Fernandes] Fixed single playing quitting not working. Re-enabled the...
 Players	Mon, Mar 12	3	[Simon Fernandes] Imported the game
 Preferences	Wed, Mar 28	24	[Simon Fernandes] Server IP is now stored in the settings cfg
 Server	Wed, Mar 28	24	[Simon Fernandes] Server IP is now stored in the settings cfg
 settings	Wed, Mar 28	24	[Simon Fernandes] Server IP is now stored in the settings cfg

Figure 24 - Subversion control log at www.assembla.com

6.3 Security

During the early stages of development, I began to wonder about what security features I would need to implement into the game. It didn't take too long for me to realise that spending a vast amount of time coding a secure game was a total waste of time.

The reason I decided not to take security into consideration much is because of the entire nature of the project. To be blunt and straightforward, making a game of Battleships secure is completely unnecessary. The game does not handle any sensitive data, nor does it contain any mission critical responsibilities. Leaving out security means in this instance means that the worst possible outcome is that a player somehow manages to cheat - not a big deal for such a casual game! Even if I did make the game more secure than it is, Java programs very easily be decompiled. A malicious user would be able to make any adjustments to any security measures I implemented anyway, so there really isn't much point.

The only place having a secure Java application is when the majority of the work is performed on a server as no-one would be able to manipulate its contents. Due to the way the game was intended to work, this simply wasn't a viable option. Lastly, making the system secure was not a part of the requirements for this project.

7 Testing

7.1 The Purpose of Testing

Testing in my opinion is one of the most important stages in the development process. It is the main opportunity for the programmer to ensure that every component of the application works as it is intended to. Testing is important because mistakes can be very costly.

While you might be able to argue that having a completely bug free game of battleships is a little overkill, I

believe it is very good practice to get into the habit of rigorously testing all your software before distribution so that if one day, you are assigned to work on a piece of mission critical work, you will have good experience in catching and fixing any problems before disaster occurs. You wouldn't want something to go wrong when developing the software for a nuclear power plant, would you?

Unfortunately, due to time constraints I did not have enough time to design and carry out a rigorous test. Instead, I decided to produce a test plan that would cover the main components of the game that should hopefully find any faults that might come up during a standard match.

7.2 Test Plan and Results

The test plan was designed to test whether or not the requirements specified earlier on in this report were met. They do not test every individual case, only the core components. The table below shows the tests that were carried out and their respective results.

Test case	Description	Data Input	Expected Result	Actual Result	Pass?
Start program	Make sure the game actually starts up properly	N/a	Game launches and shows you the name entry screen	Game launches and shows the menu screen.	PASS
Enter name	Enter a valid name	"Simon"	The main menu launches and shows you your nickname	Game launched and shows the nickname as "Simon"	PASS
Enter invalid name	Entering a name that does not conform to the allowed length.	"Si"	Display an error saying that the name is invalid.	Error showed.	PASS
Mute Music	Mute the music that continuously loops	N/a	The game stops playing the music when the mute button is clicked	Music stopped playing	PASS
Play Music	Play the music	N/a	The game plays the music when the mute button is clicked	Music started playing	PASS
Change Nickname	To change the player's current nickname	"TestUser"	The game changes the players name to "TestUser"	The game changed the players name to TestUser	PASS
Change Preferences	To change the player's visual preferences.	N/a	The game saves the colours that player chooses from the colour picker.	Colours were saved both in the settings file and across the board.	PASS

Start a single player game	To commence a single player game against an AI opponent	GridSize "12"	The game shows the ship setup screen based on the grid size specified in the prompt	Ship setup screen showed with the correct size	PASS
Place ships	To lay down your ships in preparation for a match	N/a	The game should allow the user to select the locations of all the ships and report any placement errors.	Game reported invalid ship placements and allowed me to start the game once all ships were placed.	PASS
Attack Tile	Attack an enemy tile	N/a	The game should show the selected tiles status. It should show if a tile contained a ship or not.	The game showed hit and misses appropriately. The GUI updated instantly.	PASS
Resign from match(SP)	Quit a match and return to the main menu	N/a	The game should end match and return the user to the main menu.	Match ended and the main menu was displayed	PASS
Start Multiplayer Match	Start a multiplayer match against another human opponent	grid size 15, port 2215	The game should show the ship setup screen and allow you to lay your ships. It should then wait for an incoming connection and then start the game when the opponent is ready.	Ship setup displayed and then the connection status screen opened. Nothing happened until a connection had been made. The status updated and then the game started.	PASS
Send message	Send a message to another human player	"Hello there"	The message should appear on your chat box and should appear on the opponent's screen too.	The chat log showed the messages sent on both parties screens.	PASS
View leaderboard	View the leaderboard which displays the current high scores	N/a	A leaderboard should appear showing the most up to date scores currently stored in the leaderboard database.	The leaderboard appeared.	PASS
Submit Score	Submit your score at the end of a round	N/a	The system should store the score and display it in the leaderboard when accessed	This feature was not implemented. There was no way for a score to be saved.	FAIL
View available games	To see a list of currently available multiplayer games	N/a	A menu should appear showing all of the currently available multiplayer games	A menu showed up with a single test instance of a multiplayer game	PASS

Join multiplayer game	To join an available multiplayer game	<i>Servers ip, Servers Port</i>	A connection to the server should be established and the ship setup screen should be displayed	Connection was made without problems and the ship setup screen was displayed	PASS
-----------------------	---------------------------------------	---------------------------------	--	--	------

7.3 Test Result Findings

All in all, the game functions pretty much as you'd expect it to. Unfortunately, I was unable to test score submission feature as there was not enough time for me to implement this and therefore failed the tests.

It is worth noting that these tests only show the results of expected behaviour. In reality, I should have tested every different scenario and tried to perform actions that might be considered out of the ordinary. However, it was my focus to test all of the functional requirements of the game and not every minor bug that may be present in the system. It would have taken too long and is perhaps something that I could improve on in similar future projects.

8 Project Evaluation

In this section of the report, I will be discussing the strengths and flaws of my work on the project overall. I will be looking at what I did, what I should have done and what I could have done better. I will also attempt to briefly outline tasks that I would have liked to have carried out if given the opportunity.

8.1 Design Evaluation

In the designing stage of the project, I managed to create several UML documents which greatly aided in the implementation process. I feel that the state and sequence diagram were the most beneficial to me as they provided a clear guideline on each step and process that needed to be coded. Considering what I believe to be a very short development time of only 12 weeks, I feel that I did put an adequate amount of time and effort into the designing of the game.

Out of all of the produced diagrams, I felt that the class diagram was probably the biggest let down. The original class diagram that I produced was mostly empty, and only reached the stage it is in now because I constantly kept updating it while improvising the code. Early on in the coding stage, I realised that my original class diagram made little to no sense and wouldn't in practice actually work in the programming stage. In the future, I think that producing a better class diagram could have saved me countless hours of effort during the implementation.

Out of the entire development process, I feel that I performed the worst in the games design stage. I feel I could have done further research on frameworks that would have sped up the development time significantly. Furthermore, I feel that I could have performed some visual designs and perhaps even a storyboard. Having some kind of visual designs would have given me a much clearer picture as to what exactly I wanted to have on each screen. Instead, I choose to improvise and while I'm not disappointed with the result, I am certain that continuing with this kind of practice is only going to prove disastrous in a mission critical project.

Overall, the design stage went okay, but there is definitely room for improvement. If I were to do this project all over again, I would dedicate an extra week or so to design. I would ensure that I have at least some form of visual design and I would spend time having my class diagram(s) thoroughly checked over to make sure that I did not waste any time.

8.2 Artefact Evaluation

I am very proud of my achievements with the final game. When I initially set out my goals, I did not think that I would manage to finish pretty much all of them except one - the server browser. I was honestly under the impression that I would have only been able to have complete half of the requirements that I had stated in my detailed project proposal and my requirements. I will take this opportunity to talk about what exactly I achieved out of those requirements and how I would have improved if I had the chance to do it again.

The core objectives that I set out to complete were the following:

- Produce a functional single player mode
- Produce a functional multi player mode that allowed a player to host or join a game over a network connection
- Submit and view score to a leaderboard
- Connect to a game via the use of a server browser
- Provide some form of customization

Out of all of these objectives, I did not manage to complete one- submitting a score to the leaderboard.

For the single player mode, a player is able to play through an entire match against the computer. The match is functional until the very end. If I had time to improve, I would have tried to make the AI a little bit smarter. At the moment, it only randomly picks a point on the grid regardless of its probability of containing a ship. This in turn has made playing against the computer a little bit easy; during trial runs I did not lose a single match against the computer. Perhaps an even better idea would be to have several different difficulty levels that the player can choose to play against.

Out of all of the objectives, I feel that the multiplayer mode turned out the best. Pretty much all of the functionality is there, and by the end of the development process, the vast majority of bugs had been fixed. I am most surprised with the turnout of this mode because predevelopment I thought that coding the game for network play was going to be the hardest challenge. It ended up taking me very little time at all. If I had the extra time, I would have liked to have added some 'booster' or special abilities that could make the game a little more interesting. For example, the ability to spoof the result of an enemy attack could be a great way to throw off your opponent's knowledge of where your ship positions currently are. But, like usual, time constraints meant that I had to focus on getting all of the core features implemented rather than special polishing like extra game modes.

My biggest disappointment with the final product was the redundancy of the leaderboard and it is the only objective that I did not fully complete. During the coding stage, I was unable to think of an interesting way to calculate a player's score based their performance during a match. This is most unfortunate, because I feel that a competitive edge is one of the reasons people enjoy playing games, I know I do. I left the leaderboard in though, for the sole purpose of demonstrating my technical ability to read and display information from a database through java. If I had the time, I would have most definitely loved to implement a scoring system, as it's the last objective in the list of things to do!

The one objective that I did not think I was going to have time to complete was that ability to connect to a game through the use of a server browser. When I was specifying the requirements, I was being ambitious when I thought I'd be technically able to code such a feature but I managed to get it done, and it works pretty well. While it's true that the server is very basic and offers practically no validation or security, the fact that I managed to get it working is achievement enough for me to be honest. The only improvement I would make to the server is to improve its security and I would have also have liked to have conducted some proper testing for it.

Finally, one of my other proudest achievements was the implementation of the game settings feature. Originally, I

only had some plans to make the font size adjustable but after a bit of experimentation I was able to get the grids completely colour customizable. Furthermore I was able to make these choices persistent through the use of a settings file. Unfortunately, my ambition got the better of me, and while customizing the grid colour is great, I had neglected to implement the basic features that I wanted such as font size, font colour and so on. Given more time, I would have loved to have improved the array of settings available to a user. Such settings could be switching to full screen, persisting your nickname and like mentioned earlier, a difficulty setting for playing against the computer.

8.3 Testing Evaluation

I definitely feel that the testing phase of the development process could have used a lot more attention. The testing that I carried out only really tested the main functions of the game. Could you start a new game? Could you change settings? etc. What I did not have time for however, was the testing of unexpected player input. This means that there are likely bugs in the game that are yet to be fixed, while not really that much of an issue due to the nature of the program; it would still be nice to have them all addressed.

It would have also have been hugely beneficial to have arranged for some usability testing from people who have not been involved in any stage of the development process.

9 Conclusions

9.1 Technical Conclusion

During the course of this project, I have learnt that coding in Java is fairly straight forward providing that you have made the appropriate planning and that you really think about what you're doing when you write the code. Game development in Java appeared to have been a wise choice, although I can only conclude this about 2D static games like the one I created for this project. I am currently unsure as to whether or not Java is a suitable candidate for more CPU intensive game development such as those that require the rendering of a 3D environment.

I would also like to note the importance of the Netbeans IDE for the development of the game. Not only did it make the creation of the GUI's so much easier, but it also provided appropriate tools for creating the class diagrams. Without the use of Netbeans, I am not sure I would have made it as far as I did with this project.

9.2 Project Conclusion

Overall, I feel that the project was overall a success. Other than the missed leaderboard objective, everything I set out to do got done. My biggest gripe with the project was the severe lack of time that was made available to use for completion. The allocated 12 weeks does not feel like it was anywhere near enough.

At some points during the development, I felt like I had to rush certain parts in order to try and stay on track for the April 20th deadline. As explained in my evaluation, I feel if we had perhaps double the amount of time, I would have been able to produce a polished feature rich game and would have had much more time to properly plan out all aspects of the game.

References

Notes about Netbeans code generation:

Please note, due to the nature of Netbeans' drag and drop capabilities, some of the methods presented in the source code were not hand coded by me, but rather were generated by Netbeans due to the building of the graphical user interfaces. More specifically, the method initComponents() are the auto generated methods and should not be assumed the code represented inside is of my doing.

CyberPsychology & Behavior Apr2007, Vol. 10 Issue 2, Lee, Moon-Soo Ko, Young-HoonSong, Hyoung-Seok Kwon, Ku-Hyung Lee, Hyeon-Soo Nam, Min Jung, In-Kwa - Accessed 12/04/2012 10:26am

http://upload.wikimedia.org/wikipedia/commons/thumb/6/65/Battleship_game_board.svg/220px-Battleship_game_board.svg.png - Accessed 07/03/2012 12:57pm

[http://en.wikipedia.org/wiki/Battleship_\(game\)](http://en.wikipedia.org/wiki/Battleship_(game)) - Accessed 07/03/2012 12:58pm

<http://www.squidoo.com/battleship-game> - Accessed 07/03/2012 1:03pm

<http://www.joystiq.com/2011/07/05/report-game-industry-worth-74-billion-in-2011/> - Accessed 29/03/2012 1:34pm

http://www.nytimes.com/2010/12/07/science/07tierney.html?_r=3&partner=rss&emc=rss - Accessed 29/03/2012 2:08pm

<http://www.pentadact.com/2011-05-27-what-makes-games-good/> - Accessed 29/03/2012 14:55pm

<http://www.tastyhuman.com/10-benefits-of-playing-video-games/> - Accessed 29/03/2012

<http://publish.uwo.ca/~craven/558/558era.htm> - Accessed 02/04/2012 - 13:33pm

<http://serverflux.com/systems-analysis-design/data-flow-diagrams-advantages-disadvantages/> Accessed 02/04/2012 14:26pm

http://en.wikipedia.org/wiki/Software_design_pattern - Accessed 10/04/2012 11:40am

<http://news.bbc.co.uk/1/hi/8551122.stm> - Accessed 12/04/2012 10:17am

Deliverable Assets

Greatest Battle Music of All Times: Lachrimae, Terminus , Audiomachine – Downloaded from <http://www.youtube.com/watch?v=9TRnU1rECfA> – For music playing capability purposes only.

<http://www.javazoom.net/javayer/javayer.html> - Mp3 third party library for Java. Used to implement the mp3 playing ability. This JAR file is called jl1.0.1 and should **NOT** be assessed.

Moon image : http://www.hanskellner.com/photos/2006/11/Full_Moon_Rising_DSC_0024.jpg

Battleship image: <http://bestgamewallpapers.com/files/battlestations-midway/battle-ship.jpg>

Ocean image: http://fc04.deviantart.net/fs33/f/2008/312/7/8/Dark_Ocean_by_kayooZz.png

Font used <http://www.dafont.com/plane-crash.font?text=Battleships>

All above accessed Feb 8th 2012 14:00~

Line drawing algorithm

<http://tech-algorithm.com/articles/drawing-line-using-bresenham-algorithm/> Accessed 2/3/2012. Used in drawing the ships onto the starting grid.