SEA 4: An Interactive Battleship

Game with Video Feedback

Kenneth J. Kelley

University of Alaska, Anchorage

CS 470

**Table of Contents**

Abstract 3

I. Introduction 3

II. Project Overview 4

2.1 Data Files 4

2.2 Initial Concept 5

III. Planning Process 6

IV. Project Requirements 6

4.1 Functional Specifications 7

4.2 System Specifications 8

V. System Design 8

5.1 Graphical User Interface 8

5.2 Data Structures 10

5.3 System Architecture 11

5.4 Algorithms 12

VI. Software Development Process 13

6.1 Development Cycle 13

6.2 Testing/Debugging 14

6.3 Time Schedule 14

6.4 Challenges 15

VII. Results 16

7.1 Final Program 16

7.2 Potential Refactoring 18

VIII. Conclusion and Lessons Learned 19

IX. References 20

Appendix A: User Manual 20

Sea 4: Interactive Battleship

**Abstract**

Sea 4 is an application that was created using C# and Visual Studio. It is a fully interactive version of battleship, which aims to create an immersive experience for the player. It utilizes embedded videos to provide live action feedback in response to user actions, and consists of a fully functional GUI that allows the user to easily interact with the game. A variety of features were added to enhance the game play, and enable the user to customize the game to fit their preferences. This project was developed for CS470, and provided me with additional experience regarding the software development process. From concept to implementation, the project’s design was consistently being reviewed and revised, to incorporate the feedback of my professor and peers.

**I. Introduction**

Most people can think of a time, when they were able to enjoy a classic board game with family or friends. Board games appeal to us, because they are simple to understand, but able to provide a fun, themed experience. You can be a rich tycoon, or even the commander of a ship. The game known as Battleship has been around for many years now, and most are familiar with its concept. However, with technology rapidly outpacing itself, most kids of today are looking to computers for their entertainment.

This is not necessarily a bad thing. Computers have the capability of providing a much more realistic experience to a player. Game design has become a massive industry, and today’s games are top notch. However, these classic games still hold great value, and should not be forgotten. They just need to be given the chance to adapt, and take advantage of this new technology. Luckily for Battleship, I was able to do just this.

For my semester project, I chose to tackle game design. I have always enjoyed video games, and they are one of the reasons I have chosen to pursue computer science. I wanted something that would keep my interest throughout the course of the project, allow me to be creative, and incorporate a wide variety of programming concepts. As such, I attempted to create my own version of one of my favorite board games, Battleship.

The intent was to bring the game to life, and make it an interactive experience. The game needed to be immersive, so it would have to respond to a player’s actions, and provide them with some form of visual feedback. Additionally, I wanted the experience to feel a bit different every time, so I needed to add features that would make the game customizable. It would not be an easy task, but this was to be an explosive game, and so it was given an explosive name... SEA 4!

**II. Project Overview**

This project was intended to increase a variety of programming skills. Game design was selected because it requires the manipulation of user input, tracking of data and variables, extensive use of graphics, some form of artificial intelligence, and the design of a user friendly GUI. There are many games that I could have chosen to implement, but I opted to create an interactive version of the classic game Battleship. I called my implementation SEA-4. The goal of my project was to focus on the design and the implementation of the game’s various interactive components, functionality, and critical elements. Creating an original game from scratch would have required much more attention to game-specific details, such as the rules of the game. I chose the familiar game of Battleship, because the rules are already well understood, and this allowed me to focus more on the design and code, rather than creating and defining rules for the game.

The game needed to be easy to use from the user’s perspective, so all input and output takes place through the GUI. This was heavily based on the MVC pattern. There were buttons (the Controller) to accept user input, this adjusted the game state (the Model) in some way, and the effect could be seen from the games verbal and visual output windows (the View). Since I tried to make the game more interactive, I added many “fun” features to enhance game play.

The game allows the user to interact in various ways by placing ships, adjusting game settings, and choosing firing coordinates. The user is rewarded with both verbal and visual output, depending on their actions, which includes actual embedded video of ships being hit or sinking. The game’s setup phase is instructive, and guides the user on which actions need to be taken. Finally, the game also provides a menu that is used for setting game options, creating a new game, or saving the game state to a file so that it can be loaded at a later time.

*2.1 Data Files*

My application relies on several file types. Some are are used to implement the interactive features, but others are created by the program itself. The important file types, as well as their function, are described below.

1. txt - The program provides the ability for the user to create game state files to save a game. These files are later used to load a game state, so that the player may continue where they left off. These data files store information about a game state, and that information is used to change the state of the GUI, to represent the saved model and view.

2. jpg – The game utilizes many images. To make the game more visually aesthetic, the placement phase features images of each current ship, along with a description of the ship. The pieces themselves are images that are superimposed over the buttons where they are placed. Additonally, the classic battleship diagram, which indicates the hit points of each ship and is featured on the board game, is also included.

3. wmv -These files are windows media video, and contain real life video footage. There are two essential video files that are used. One shows a ship being hit, and another a ship sinking. They are both used to enable visual feedback when a ship is hit or sunk. A windows media player is actually embedded into the application, and contained in the action window. A video is played at different times depending on the user’s actions, which helps to provide the user with a more immersive feeling, and feel like they are playing a “real” game. The files themselves come from government file footage, of decommissioned ships being intentionally sunk. The soundtrack for the game, also relies on this same mechanic, except the video is looped, and not visibile.

*2.2 Initial Concept*

The initial concept of the GUI for the game is shown below. The final product indicates that the design stayed relatively true to its initial form. However, it did change in several ways to address different issues as they were encountered. The differences can be identified in the GUI for the final product featured in chapter 5.1, but most noticeably is the use of ships for pieces.



**III. Planning Process**

This project was selected because it was a personal interest of mine. Designing video games is fun and this made it easier to work on, but getting started is always a difficult challenge regardless of the nature of the project. Since there was not an existing code base, my application had to be designed and implemented from the ground up. This meant that much of the planning needed to occur before I could start implementing anything.

Initially, I tried to plan out all of the project details during the pre-proposal. I had decided how many hours I would be able to allocate each week, and what I would try to work on. However, many challenges arose during the actual implementation of the project. This required me to make modifications to the design, and restructure the schedule to ensure that I would still finish on time. Even though I was very thorough in my initial design, the planning phase tended to occur throughout the length of the project and ended only when the project was finished. This was not really a major issue though, because the project was unique in one major aspect.

I was the client. I was not building the application for anyone else, and so I did not have to immediately justify the changes I decided to make. This provided me with some advantages, but also some disadvantages. Since I was the client, I was afforded the luxury of having total creative control. This meant the project was more flexible, I could add more features, and decide on better strategies for implementation. However, the cost was that I had to play both roles. I had to personally create the user stories and requirements, and this meant that I had to decide if there was something wrong with a particular part of the program. A client is able to tell you if they like something or if they don’t. Criticizing your own work is tougher challenge, and I constantly found myself trying to decide if I should have done something in a different way. This resulted in more frequent changes to the design, and a lengthier planning process.

**IV. Requirements**

The requirements for Sea-4 had to be thorough enough to allow an entire game of Battleship to be played. Game settings needed to be customizable via the menu. This would allow a player to tailor the game to their preferences by selecting the number of ships, changing the difficulty, and turning the music on or off. Once the game settings have been established, there needed to be way to start the game. This was to be in the form of a button.

However, before a game could be played it needed to have a way of being setup. This meant a player needed to be able to have a way to place their ships, the number varying depending upon the options. The computer would need to place its pieces, depending upon the difficulty. Both of these requirements lead to the storage of game state data, creation of data structures, and the initialization of an intelligent AI opponent.

Game logic also needed to be in place, in order to restrict the user from making illegal moves or ship placements. For example, a player cannot select a previously targeted position. The interactive component would directly influence the design of many components, particularly the GUI. The GUI was responsible for regulating user interaction, via phases, and displaying results of a user’s actions. It needed to inform the player of changes to the game state via the sea-grids, message box, and action window.

*4.1 Functional Specifications*

1. Game Logic – Must be able to detect valid moves, and ship placements. The number of valid moves will decrease as the game progresses, and there are fewer positions to attack. A ship should not be allowed to be placed if it will overlap with another ship, or go off the grid.

2. Game Data – Application needs to maintain data about the game state. While a game is in progress, both computer and player ship grids should have their current states stored, and updated via game actions. If a game is to be saved, and loaded at a later time the program needed a way to export a game state to a file. Text files would store the game data for later use (loading a saved file), and the application would need to understand the way they were organized. Arrays would be the most useful data structure for storing and manipulating the game data.

3. AI Opponent – An intelligent AI player of varying difficulties. Easy mode would randomly choose firing locations. Normal mode would randomly choose until it found a ship, then it would attempt to find the ships direction and continue sinking it. Hard mode, would only allow a certain number of misses before the computer found a ship. Lastly, impossible mode would find each ship successively. The AI must not make invalid moves, and is also responsible for the random, but valid placement of its ships.

4. GUI – Interactive user interface that will accept all user actions, and respond appropriately by displaying changes to the game state, and informing the player of the result of their choices. It is based on a MVC pattern.

5. Menu - Allows the user to select choices such as Exit, New, Save, and Load, and Options such as number of ships 1 – 10, difficulty, and music.

6. Load/Save – Load and save a game state to and from a text file.

7. Dialog Box – To verbally inform the player of the results of the game.

8. Action Window - Visually informs the player of the results of the game, through the use of embedded videos. It is also used during the placement phase to provide an interactive walkthrough for the player, displaying the type of ship to be placed, and providing a description. This is the primary interactive window, and additionally, it displays a welcome screen while a game is not in progress and indicates when the game is over.

9. (2) Sea-Grids - One for firing and displaying results of a players actions, and the other for setting ships and displaying results of computers actions.

10. Ship Counter – Keeps track of the number of ships left for each player.

*4.2 System Specifications*

The system was constructed using Microsoft Visual Studio 2010. There is also an installer included with the program that will allow a user to install the application to a folder of their choice. However, it does not place a shortcut on the desktop. The application is written, using the C# programming language, and the operating system used was Windows Vista. Since the program is entirely designed from Microsoft products, a windows environment is considered to be the most optimal when running this program. The screen resolution was set to 1400x900, but the window size of the program is small enough to display on most computers. For best results, these specifications should be the minimum requirements, but the game should be viable on other systems.

**V. Design**

The application was coded using the C# language programming language, and the Microsoft Visual Studio 2010 IDE. C# is an object oriented language, and the design consists primarily of an instance of the main form, which displays the GUI. There are many actions a user can perform through the GUI, and the appropriate methods for each action are found within the main form class. Several other classes are responsible for maintaining the game state, enforcing valid moves, and creating the AI component.

*5.1 Graphical User Interface*

The interface was designed to be easy to use, but thorough enough to meet the requirements of the game. It possesses a menu, which allows a user to adjust the game settings (AI difficulty, number of starting ships, music), and create, load, or save a game. It also has several important buttons. Some of the buttons only become visible when the game is in a certain state. The start button is the most prominent and it is used to start the game. Once a game has been started the start button is replaced with a ship counter, indicating how many ships each player has remaining. A classic battleship grid can also be found on the GUI, which indicates the number of hits required to sink each type of ship.

There are two ways in which the game can communicate with the user. One is through the dialog/message box. This box displays the game progress, and verbally informs the user of the results of their actions. If a game is not currently underway, it provides instructions on how to start a game. The other method the game uses to provide feedback is through the Action Window. This window is highly visual and attempts to make the experience more immersive. It displays images for the welcome screen, as well when the game is over, but its main appeal is evident during the game play.

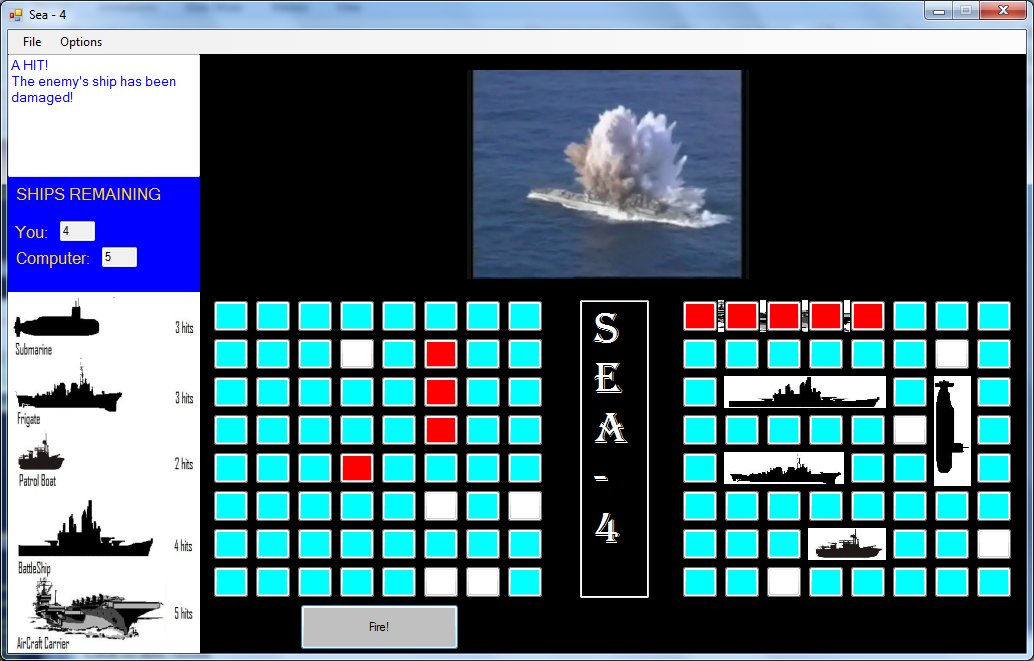
Even during the setup phase, the Action window is able to provide the user with a better experience. It indicates which ship currently needs to be placed, by displaying an image of that ship, along with a brief description. The images change to the next ship, as each is placed on the board. However, the most notable feature of the Action Window occurs when the player hits or sinks a ship. The Action Window contains embedded videos, which depict real life footage of a ship explosion, or a ship sinking, when the user scores a hit. The music for the game is implemented in a similar fashion, but is always playing the background.

The image below demonstrates the action window during the ship placement phase.



Lastly, the GUI contains some important input controls. There are 2 “Sea Grids” that the user can use to interact with the game. The grids are each composed of an array of buttons, and the buttons for each grid call the same methods, only with different values. One grid can be used to fire upon the computer’s ships, and it displays the hits or misses of the player. This is referred to as the Firing Grid. Once a location on the firing grid is clicked, it highlights gold. The user may then click a button below the firing grid, that confirms the location, and fires.

The other grid, called the Ship Placement Grid, is used by the player during the ship placement phase. The user may select a position, and the grid will display the possible ship layouts for that position in red. The user then clicks a ship layout to select it, and finally clicks the “Place Ship” button to confirm their choice. They may cancel the placement at any time, by using the cancel button. Once all ships are placed on the grid, the game starts, and the grid becomes locked. However, it is used during the game to indicate the hits and misses of the computer, and shows the status of the player’s fleet.

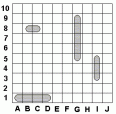


The GUI example above displays a game that is currently in progress. The player just scored a hit, and this can be seen in the Action Window. Additionally, this hit is stated in the dialog box. The two grids are both being utilized. The grid on the left is the firing grid, and shows the players hits/misses. The grid to the right is the ship placement grid, and the player’s ships can be located on the board.

*5.2 Data Structures*

The data structures used in this project were the ever useful 2-dimensional arrays. This seemed like a great choice, because the game board itself looks like 2 arrays, so the transition from a game state to a data structure could be easily mapped. The key data structures for the system were two 8x8 arrays, (example state is 10x10) but they were used as a means to many ends. They were mainly used to store the locations of both the player’s and the computer’s ships. However, they turned out to be essential during the ship placement phase, and game phase.

During the ship placement phase, they were used to determine valid positions, of both player and ship placement algorithms. If a ship was successfully placed, then the arrays were updated. When a new ship was to be placed, the algorithms had to check the locations in the array to make sure the space was available. During the game phase, the arrays were updated based upon the results of the game. The state was always current, and if the user opted to save the game, the states of both arrays were written to a .txt file. The state from a file could be loaded at a later time, and the program would initialize both arrays (computer and player) based upon the game states of the file. Basically, the array data structures were at the heart of the game logic, and greatly assisted in the functionality of the game.

A game state (array) can be observed below. The two digits are appended when saving to a text file. They indicate the number of ships remaining.

(Game State Key)

XX – open space MM – Miss HH – Hit

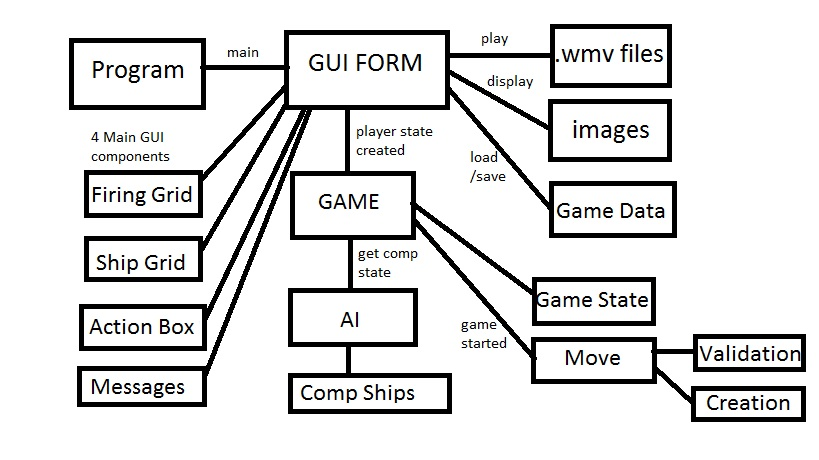
A1 – A letter followed by a number indicates ship type, and number. A1 is Aircraft Carrier 1.

Additional Codes: S – submarine B – Battleship F – Frigate P – Patrol Boat

*5.3 System Architecture*

The system consists of several classes and files, which are all used by the main form. The main form displays the GUI, and it is responsible for taking the correct action when a user makes a selection, based upon the current phase of the game. This action can be one of many things, such as playing files, or creating instances of the other classes.

The relationship between the classes can be identified in the following figure.



The System Architecture

(Overall design, and the interactions among various elements.)

*5.4 Algorithms*

The game relied on several algorithms, most of which were concerned with the AI, or actions of the sea grids. The AI had several tasks that it needed to complete, and the algorithms that were used were able to produce an acceptable response to each challenge. The tasks for the AI involved the following:

 Setting the computer’s ships, making random valid placements.

 Selecting a valid target to attack, based on the difficulty settings selected.

 Continuing to sink the same ship if one is found, or trying locations near the ship.

There also needed to be algorithms in place to initialize the game state, and validate the actions of the player. The following challenges were addressed:

.

 Allow the player to place ships, while systematically eliminating possible positions once a ship is placed.

 Initializing both firing, and setup grids to empty values.

 Translating save data to a game state when loading a game, and changing the state of the GUI to represent that loaded game.

 Translating a current game state to data when saving a game, that can be stored in a text file

Additionally, algorithms were responsible for some of the game logic.

 Alternating between computer and player, and displaying results depending on hit/miss

 Evaluating win/loss condition

 Changing the game phase when certain conditions are met

Many algorithms were designed to achieve these goals. However, the trickiest algorithm to implement was actually concerned with the placement of ships by the player. The initial algorithm became bloated quickly, and to address this I had to divide the ship placement phase into sub-phases.

First a player would pick a position, and then the algorithm would verify it was a valid position before highlighting the possible layouts, and shifting to the second phase. During phase 2, another algorithm would allow the user to pick a location that was highlighted red, and it would ensure that only that position was highlighted once selected, by systematically coloring any positions that ships did not already occupy back to the ocean color, it then shifts to phase 3. Phase 3 required a third algorithm that allowed the user to confirm the choice, then makes changes to the game state, and places an image of a ship over the buttons.

**VI. Software Development Process**

*6.1 Development Cycle*

The fun thing about this project was that I got to see it evolve from the concept to the final product. The development cycle that I experienced was very close to the waterfall model, in that it had a requirements phase, a design phase, an implementation phase, and a testing phase. I did everything in that order, mostly to adhere to the class assignments, and deadlines.

I was required to produce a pre-proposal, which essentially identified the requirements of the project, and described a basic design. The design phase continued until after the first presentation, where I was able to incorporate the feedback of peers (such as ship images for the pieces). Each phase was mostly completed before I moved onto the next, and by the time I got ready to implement the project I thought I had a fairly decent design. The implementation phase was initially slow moving, but I picked up speed, and was able to have a mostly complete working model to begin the testing phase. I personally tested most of the application myself, but allowed a heuristic evaluation to be performed by my peers. This was helpful, and alerted me to other issues to consider. Although this model worked well, I did deviate from the traditional waterfall model in one major way, which is addressed in 6.4 challenges.

*6.2 Testing/Debugging*

When I tested the program, I mostly used black box testing. I approached it from the perspective of the user, and intended to “break it”. I mostly tried making invalid moves, and incorrect actions, to see if the program would catch the errors, and take the appropriate action. I also made some valid moves, but mostly tested these types of moves after I had implemented a certain feature in the program.

Once I had a working game. I made many test runs by playing the game through from setup to finish. To assist me in this endeavor, I actually made a cheating method that would allow me to see where the computer placed their ships. This allowed me to target a certain location, and have an expectation of what the result should be. I used this strategy to debug most of the game logic. This process continued for the different features, and game settings, until I was satisfied that I had found most of the errors.

I also relied on the reviews of my peers. They were able to find a list of “defects”, though some were more about preference than being defective. I did not have time to address all the issues, and most were small or would require a major refactoring of the design. However, I did take care of a few that either would enhance the game play or addressed a major issue that could be fixed without restructuring most of the code.

The sea-4 title could initially be changed by the user. I have disabled this, so that the player can no longer change the name of the game. This was not desirable. I also added a delay when the player misses, and it indicates the computer is “thinking”. This allows the user time to see where the computer moved. I did not do this for a hit, because this allows the player to try another space more quickly once they have found a ship.

The player is now notified when a hit is achieved, by an embedded video. Additionally, I also added a verbal message for the action window that displays an italicized “MISS!” so that both results are accounted for. The size of the window frame was also scaled down, and components were shifted in the GUI, to allow the game to fit more easily onto a smaller screen. The only major issue I was unable to address, was adding ship images to the game board when a game is loaded from a save state. This was due to lack of time.

*6.3 Time Schedule*

I originally estimated that I would have 8-12 hours a week to work on this project. While this was a good estimate, it is probably more indicative of the average time I spent working on it, rather than the time spent per week. The truth is, that I probably exceeded 12 hours towards the midpoint of the project, while working less than 8 in the earlier months. This lead to a code-like-hell approach at times, and it was mostly attributed to problems I encountered with the initial schedule. This initial plan detailed all the tasks that would need to be accomplished and allocated a certain amount of time to each, as well as a start date. However, I ran into problems almost immediately.

It turned out that I had not accounted for the priority of tasks. One of my first challenges was to implement the video portion of the program. The goal was to display hits and misses, and it was discovered that this was not as important as identifying hits and misses to begin with. In order to create a hit or a miss, I would need a working game. I continued working backwards, realizing that to have a game, I would need a game state, and to have a game state, I would need a way of initializing the game. This lead to the restructuring of the entire schedule, and the first task became the initialization of the game, with the focus on letting a user interactively create a state by placing ships.

There were other changes I made to the design, which made the initial schedule even more useless. For example, I decided to abandon the initial concept of using DirectX all together. This caused me to look for alternatives during time that was originally intended to research DirectX. Lastly, I added many features as the game was being implemented. This caused me to add more tasks to the schedule that were not initially taken into account. In short, I relied more on deadlines, and implemented features as they were required. Implementation followed a logical progression. For example, after I was able to establish a game state, I worked on the game logic. Once I had a working game, I then began to add the extra features, such as the video display.

*6.4 Challenges*

Waterfall Model – I ran into some problems adhering to the waterfall model. I found myself jumping back to other phases after I started the implementation. Certain design decisions turned out to be less than optimal, such as my initial decision to use DirectX, instead of Windows Media player for embedded videos. I also tried to add additional features, to make the program more interactive, and this caused me to return to the design phase quite frequently.

Peer Feedback – I found it difficult at times to fully accept other people’s ideas. Some ideas were a matter of preference, such as removing the firing button from the game. I had to choose which ideas I felt would help the program, and which I felt would hinder it.

Expecting the Unexpected – I found it challenging to try to address every possible user action. I approached this challenge, by designing a phase system. Certain parts of the application would only become accessible when the program was in a specific phase. A phase could only be attained when certain actions were conducted.

Time – Trying to add different features to the design and make the game more fun, but still ensure that I would be able meet project deadlines.

**VII. Results**

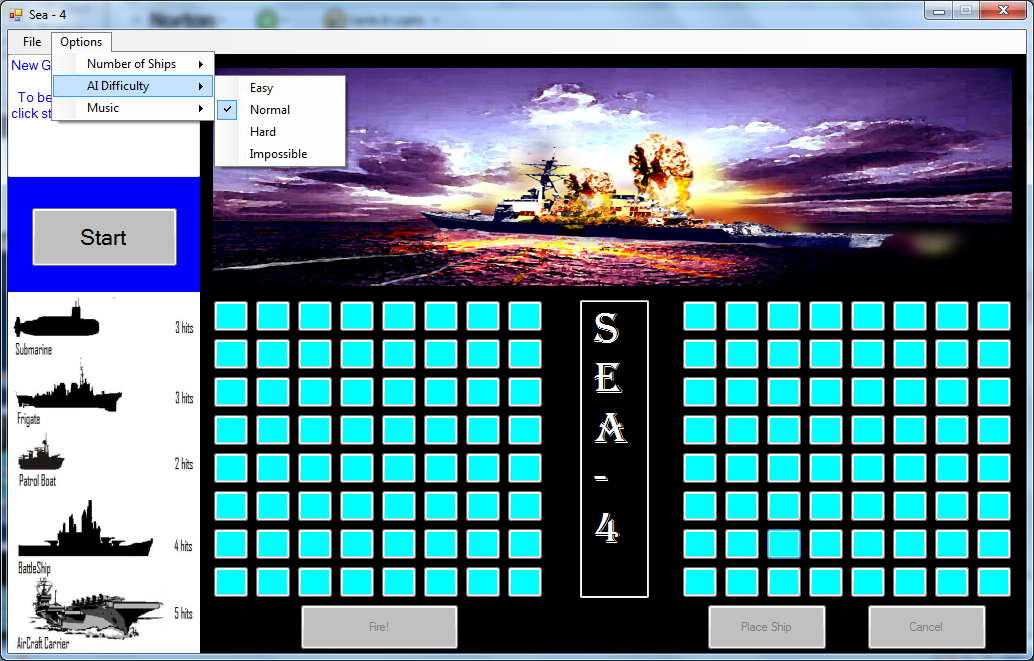
Sea-4 was completed on time, and was successfully implemented. It is fully functional, and addresses all the requirements that I had originally established. A user is able to experience an immersive game, with an easy to use GUI. They can interactively set their pieces, choose firing positions, and customize the game settings. The program rewards the user with visual feedback in the form of real life video footage of naval ships, and provides a fun gaming environment. Sea-4 has the “feel” of the classic naval warfare game of Battleship, which it is intended to emulate, and it brings the game to life.

As I was the client, I was happy with the final product.

*7.1 Final Product*

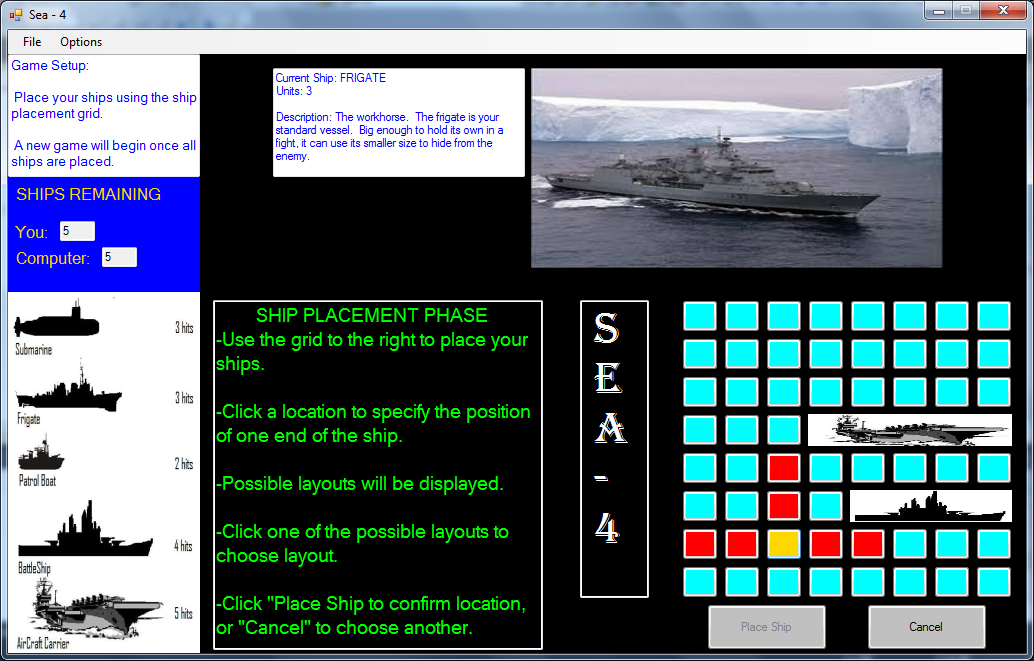
Sea-4 screenshots can be seen below. Each one shows the game in a different phase. For a full explanation of the program features, please see the User Manual (Appendix A).

START PHASE



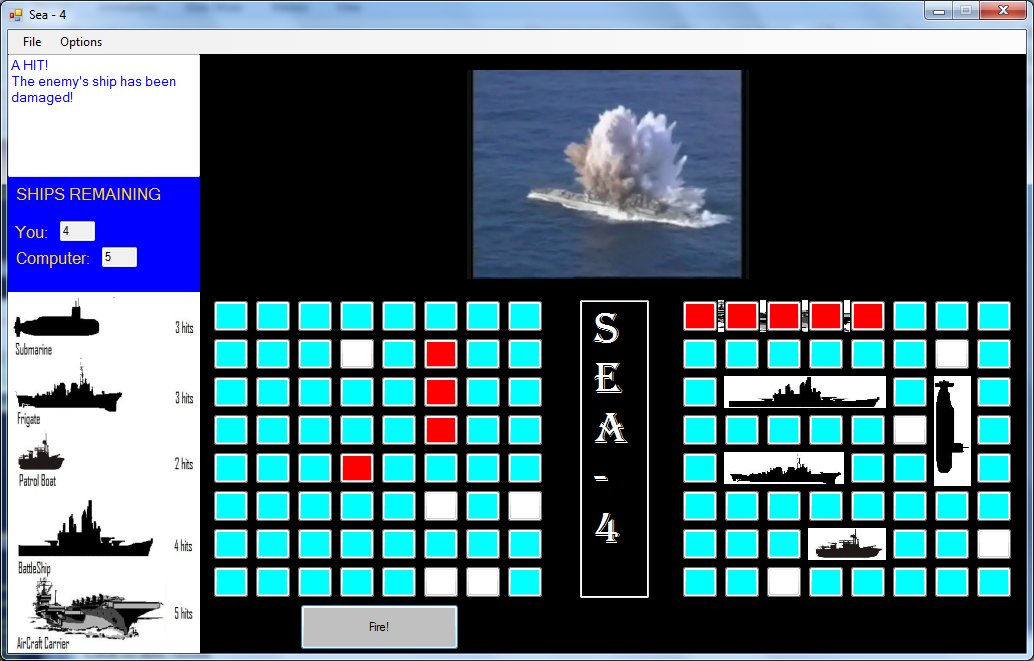
The start phase allows a user to start a game by clicking on the start button. They may adjust game settings at this time (ships, AI, music) to customize the game. The Action Window displays the games welcome screen, and most of the buttons are inactive.

SHIP PLACEMENT PHASE



The ship placement phase walks the user through the placement process. They can use the right sea grid to select a position. This is highlighted gold, and possible layouts are displayed. A user can then select a layout, and place the ship on that location. In the image, 2 ships have been placed, and the current ship to be placed is the Frigate (unit 3), indicated by the Action Window.

GAME PHASE



The Game phase is the actual game play. The player may choose a position on the left grid and click the Fire button below. If a ship is hit or missed it is indicated via the Action Window. In the image a hit has resulted in video feedback. The computer targets the player’s ships via the ship placement grid, and in the image it has sunk the player’s aircraft carrier. Lastly, a ship counter displays the remaining ships of both players.

GAME OVER



Once the game is over, the game over phase gives the player the option to play again, and indicates that the game is over. This phase will transition back to the start phase, and the cycle can be repeated.

*7.2 Potential Refactoring*

I was happy with the overall outcome of the project. The game is fully functional, and to the best of my knowledge, bug free. That being said, there are some things that could be added in the future.

A few of the ideas I received were great ideas, but would require significant restructuring. I believe that in the future, a top down view displaying vertical ships would be a great addition. One might also consider making the ship placement phase mouse-over capable, essentially allowing a user to move the mouse over a position and have the layout displayed for that location. I believe adding sounds to some of the buttons, for example when a user selects fire they here a rocket launch, would increase the immersive feel of the game.

The beauty of Sea-4 is that although the game is complete, there are always things that can be added. However, I would leave some of the core game code alone. The application was built from the ground up, as explained earlier, and some of the features were designed to work well with previously implemented features. Therefore changing things at the heart of the game, such as the phase system, could have a negative impact on other parts of the program.

**VIII. Conclusion & Lessons Learned**

*Conclusion*

Throughout the semester, I worked on a project called Sea-4. Sea-4 is basically a video game and it is heavily modeled on the classic board game of Battleship. I chose the path of game design because it provided me with additional experience regarding many different programming concepts. Equally important, is the fact that designing video games is fun. Since I enjoyed working on Sea-4, it was able to hold my interest throughout the entire length of the project.

The goal of Sea-4 was to take the game of Battleship to the next level. I wanted it be an interactive and immersive experience. I accomplished this in several ways. I opted to design a GUI that embodied the traditional feel of battleship, but was capable of supporting many features. It was easy to use, and players of the classic game would be able to recognize several key components, such as the 2 sea grids. The sea grids were very important, because they were used as the primary means through which users interacted with the program. They allowed the user to fire upon ships or place their own, and displayed the current progress of the game.

The immersive aspect of the game came in the form of the action window, and customizable game options. The action window displayed video footage of ship explosions, when a user scored a hit or sunk a ship. The action window also served as a ship placement phase walkthrough. It showed images of real life representations for the ships, as well as a short description for each. The dialog box, also intended to make the experience more enjoyable. It provided game dialog that attempted to make the user feel like they were the captain, and the game was awaiting their orders. Lastly, a user could choose between different options (music, number of ship, AI difficulty) , and adjust the game to fit their play style.

*Lessons Learned*

This project provided me with a lot of experience with software engineering. Since the project was built from scratch, it required me to utilize each of the development phases to ensure that the project was moving along at a decent pace. As such, I developed a greater understanding of each of the phases, and the process as a whole. I also realized shortly into the development of Sea-4 that the rigid waterfall model is very hard to adhere too. I found that a bit of flexibility is a welcome thing, and it is often necessary if you have not had years to design something.

The project also increased my programming skills. I used C# and visual studio for the second time, and they are quickly becoming my favorite language and IDE. Using this combination, I have been able to develop very nice GUIs that can support many user actions. I was also able to embed videos into the application. This was the original idea I had for the game, but it was my first time doing something like this. I learned that there are tools provided by visual studio that allow you to embed videos in your programs, but also in other things like web pages. And of course, I was given more practice with essential skills. Implementing the game logic itself, provided me with practice using IO, Data structures, artificial intelligence, and controlling the flow of the program.

**IX. References**

There was only one source that I really relied on. It addressed all of my concerns related to using visual studio, which pertained to implementing various features. In particular, I received a lot of information on how to embed a video in an application from the website. The website did not have an author, but the source was reliable, as it came from Microsoft.

- msdn.microsoft.com

**APPENDIX A - User Manual**

**Minimum System Requirements**

The system was constructed using Microsoft Visual Studio 2010 and C#, on a Windows Platform.

Operating System: Windows Vista/XP/7.

Memory: 512 MB

CPU: 1 GHz

Screen resolution: 1024 x 800

**Installation**

There is an installer included with the program that will allow a user to install the application to a folder of their choice. They may then run the executable file from that location. However, it does not place a shortcut on the desktop. Optionally, a user may run the program using visual studio to open it.

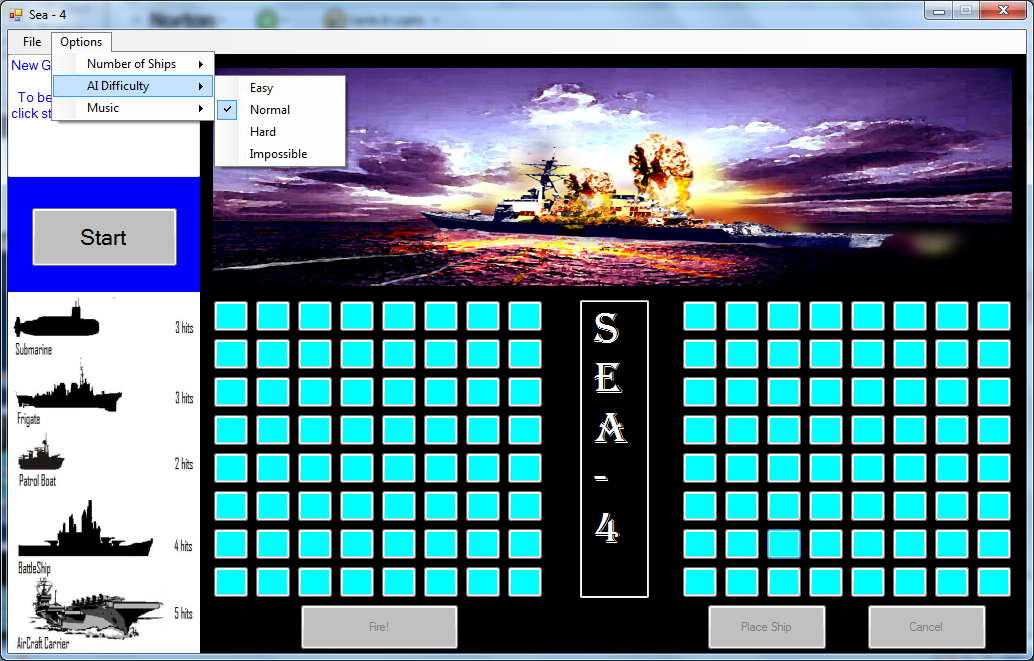
**Selecting Options**

With the exception of the music, a user may only select game options before a game has been started. Clicking on the options tab, allows a user to set the following:

Number of Ships: 1-10

Difficulty: Easy, Normal, Hard, Impossible

Music: On/Off



**Starting a New Game**

Click on the start button, in the blue square, to start a new game. Clicking this will take you to the ship setup phase. Alternatively, a user may start a new game by clicking “New Game” in the file menu. If a game is in progress, the file menu is the only way to start a new game.

**Saving a Game**

If a game is in progress, a user may save the current game, by clicking File > Save Game. They must then name the file, and that file can later be used to load the game, and continue play.

**Loading a Saved Game**

By clicking File > Load Game, a user can select a saved game file. Once selected, the game will load the state of the file, and a user can continue where they left off.

**Setup: Ship Placement Phase**

To place your ships, use the Ship Placement Grid. During the placement phase, it is the only visible grid. It consists of many blue buttons, and you will find two buttons below it, called Place Ship, and Cancel. The current ship to be placed, is indicated in the Action Window.

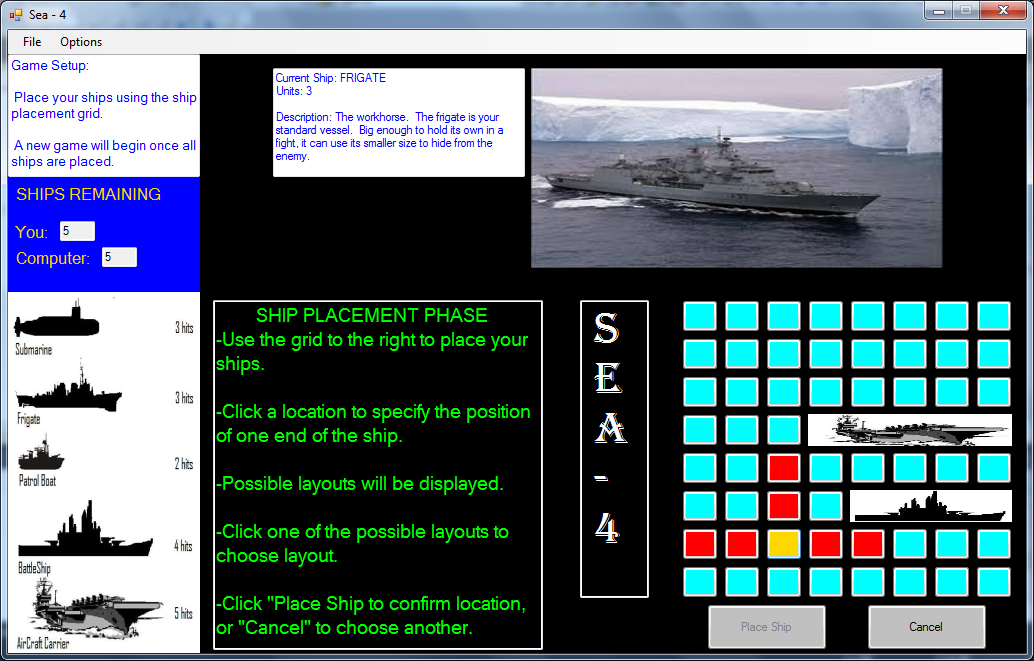
First: Click a location for one end of the ship. This will highlight that position Gold, and show the possible layouts for the ship in red.

Second: Click on one of the red layouts. This will “select” it, and the other layouts will disappear.

Third: Click “Place Ship” to confirm the location, and begin placing the next ship.

Canceling: At any time during the placement phase, you may choose another position for the current ship by clicking “Cancel”. Warning: you can NOT undo the position of a ship that has already been confirmed.

The ship placement phase is exhibited below.



**Playing the Game**

Once the ship placement phase is complete, a game will be started. The player always goes first. To fire upon the computer, you will use the Firing Grid. This is located to the left of the ship grid that was used to place your ships.

FIRING: To fire, click on a “blue” location in the firing grid. This will highlight that location gold. If you change your mind, you may click on another location. Once you have selected a desirable position, click the Fire Button, below the grid, to confirm the choice. This will fire upon the enemy ships. The result of your actions is indicated on the grid, by the dialog box, and action window.

The computer will then fire back upon your ships, by targeting the ship placement grid. You will see the results of his actions indicated by marks on the grid containing images of your ships.

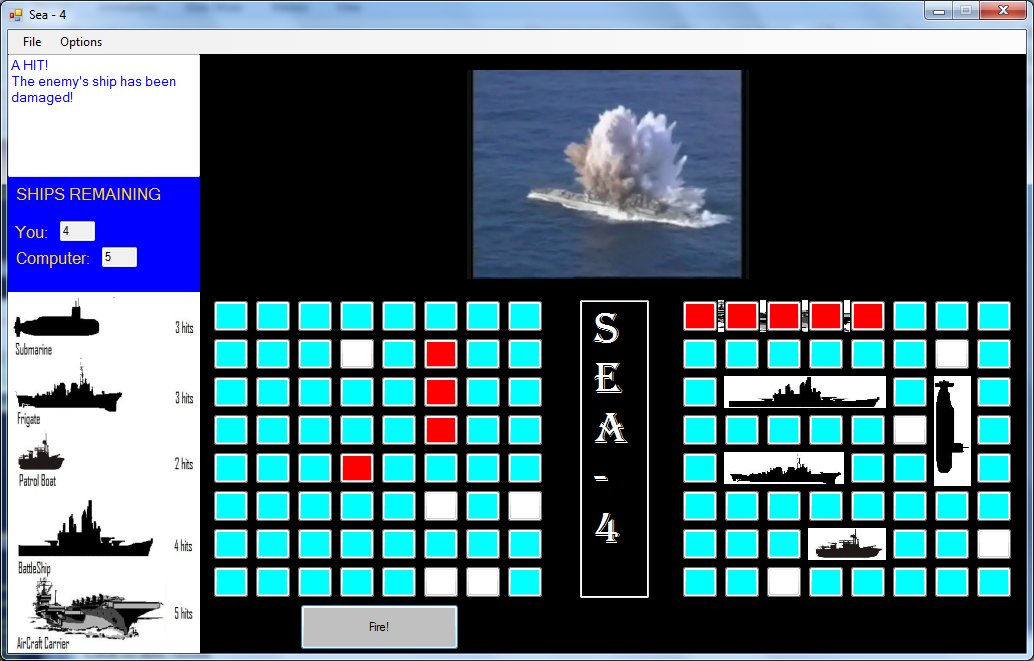
Play continues in this way until one player has lost all their ships.

**Missing**

The game will mark a miss by coloring that location White. It will also indicate the miss by displaying Miss! in the action window.

**Scoring a HIT!**

The game marks a hit by coloring that location Red. Additionally, video footage of a ship exploding or sinking can be observed in the action window. Hits/Misses can be seen below.



**Ship Counter**

During a game, a ship counter is displayed in the blue box. It indicates how many ships each player has left, and is updated as the game progresses.

**Game Over**

If a player loses all their ships, the game is over. The action window, and dialog box indicate that the game is over. The player may choose to play another game by clicking on “Play Again”.



**Screen Shots**

To get a screen shot of the program, hold Alt + PrintScreen. This will copy the program window to the clipboard. You may then paste the image into a paint program such as Paint or Adobe Photoshop and edit/print the image.

**Exiting**

When you would like to exit the application, choose File > Exit or click the X in the upper right corner.