SPECIAL THANKS

We would like to express our thanks to ***Dr. Liam Peyton*** for introducing us to this particular aspect of Software Engineering.

We would also like to express our thanks to ***OUR CLIENT*** for giving us the opportunity to work on this extraordinary project.

REPORT SUMMARY

This section is intended to be a summary of the present document. It is dedicated to give a general overview of the topics discussed in this requirements document. The more zealous reader will find detailed information about these topics later in the document, and references to the later pages will be made where needed.

1. General Description

This software is designed to be a game played by essentially the gamer community. It consists of a two-player game where the player would be the head of a battleship fleet, with the goal of destroying all his enemy's ships. It will be real-time and the viewing will be done via a camera that can be set at any height level (the player would be able to see the whole game area from a number of possible ways, ranging from a top bird's eye view to the bridge of the ship. It will be designed in a client-server fashion, where one player will be designated as the server, and the other as the client.

#	Requirements	Ve	Va
1.1	A Client/Server architecture is used in this software		

Ve: Verification Check
Va: Validation Check

2. Requirements

#	Requirements	Ve	Va
2.1	This game is played by 2 player		
2.2	Real-time game *(not turn-based)*		
2.3	Players are represented as battleships navigating on an aquatic battlefield		
2.4	Each player controls a fleet of 5 battleships		
2.5	Top View will be from top *(this is a bird-view of the sea)*.		
2.6	Firing View will be first-person, from behind the battleship's canon		
2.7	View will switched from the Top View to the Firing View when player presses the TAB button when a boat is selected		
2.8	The player exits the Firing View and goes into the Top View by pressing the TAB button		
2.9	The Top View will permit the user to move in different area of the map by moving the mouse on the border of the window		

#	Requirements	Ve	Va
2.10	In the Top View, the user will only be able to control the direction and speed of its own battleship.		
2.11	In the Firing View, the user will only be able to controls the canon.		
2.12	Current of the ocean will be simulated and boats will drift accordingly		
2.13	Wind will be simulated and will affect the boats and projectiles		
2.14	A player must be able to host a game on a LAN *(For server)*.		
2.15	A player must be able to connect to a server on a LAN *(For client)*		
2.16	Players must be able to terminate a session		
2.17	If a player terminates a session, the other player must be notified		
2.18	The map will be covered by a "fog of war" to represent the idea that on a boat, the viewing distance is limited		
2.19	Every object, except the player's boats, in the fog of war is invisible		
2.20	Each of the player's boats will be surrounded by a circular viewing area with a radius equivalent to one boat-length. Any object present in this viewing area will be visible to the player.		
2.21	Every boat is provided with a radar that can view up to radius of two and a half boat lengths from its current position		
2.22	A context menu will be present in the bottom left corner of the map at any given view. Its contents will be a Speed indicator, the Heading Indicator, Boat Coordinates, Boat Damages, Canon orientation and Power.		
2.23	The player aims using point-and-click functionality in the Firing View		
2.24	The player fires using the Space Bar		
2.25	Pressing the ESC key will pop-up a dialog in the center of the screen asking the user if he/she wants to quit (Yes or No)		
2.26	The Quit Dialog will not stop the proceeding of the game, but the user will not be able to control anything else while this dialog is up.		

Ve: Verification Check
Va: Validation Check

3. Use Case Models / Features of System

#	Use Cases	Ve	Va
3.1	A player must be able to host a game on a LAN *(For server)*.		
3.2	A player must be able to connect to a server on a LAN *(For client)*		
3.3	User can control boat direction in the Top View		
3.4	User can control boat speed in the Top View		
3.5	User can aim artillery in the Firing View		
3.6	User can fire artillery to enemy ships in the Firing View		
3.7	User switches from the Firing View to the Top View		
3.8	User switches from the Top View to the Firing View		
3.9	User can terminate the game in both of the View		

Ve: Verification Check
Va: Validation Check

TABLE OF CONTENTS

LIST OF FIGURES

OVERVIEW

CONTENT DESCRIPTION

This document is a specification document, which is the written version of the understanding of what the user wants. More specifically, it's a form of contract between the constructor of the system and his customer; written in terms that both the customer and the programmer can understand. This document is representing a complete listing of everything the client expects the Wave War game will do. This document will enables programmers to construct the exact system the client ordered. In conclusion, it represents an understanding between the client and the developer of what the customer needs or wants in a technical way, and it is usually written jointly by the both parties.

WAVE WAR DESCRIPTION

This section of the document is to give the reader a quick overview of the game that is to be developed. Therefore, it explains some of the more important functionalities in no great detail, only to give general insight on how the game is to be played and what it will look like. To read a more detailed and technical account of the requirements, please refer to the Requirements section.

Wave war consist of game that will be played by two players on different computers. The communication will be provided by network functionality (LAN). This is a game of a real time naval battle. Users would be represented as battleships navigating in an aquatic battlefield, where the goal is to hit another ship with the ship's artillery. However, the opponents' ships would also be moving, wind, gravity, current would also be accounted for, creating more challenge for the user to get his shot right. The user will have to calculate exactly how to shoot, and plan is strategy before hand.

The Wave War game is not a turn-based type of game. It is played in a real-time environment, meaning that the players all play simultaneously instead of one after the other. The maximum number of players that will be supported in the first version will be two. The design should be such that if a second version is developed, more players will be supported, with the target maximum being 32. For the first version of the software, players will have a fixed amount of boats on their fleet, namely five. Boats will have the same specs. If a second version is developed, it will provide a functionality to configure this number. This version will be developed with this requirement in mind, and this version's design will reflect this requirement.

The main components of the graphical environment are a Map, a radar, and other control devices that will be described later in this document. The Map will not be entirely visible to the user on his monitor; rather, he will have to use the radar to see beyond the limits of the scope. It will represent a space twice and a half as big as the displayed section. The radar will show where other boats are positioned relatively to the user's selected boat. All action will take place in a deep-water

environment, and the map will have no islands, only sea. There won't be different sea configurations, such as sea current. The strategy of the game is only seek-and-destroy, and the winner is the player who has at least one boat left, with all the other player's boats completely destroyed.

The game will be played using a keyboard and a mouse. It will not support the joystick. The keyboard will be used during the game to select the parameter to change. For example, by pressing the 'D' key and click on the Map, the user will be able to define the direction of its boat. By using the '+' and '-' keys, the user will also be able to control the boat's speed. Other similar key/mouse combinations should be available to select the projectile's initial power, direction and angle of attack, for example.

This game will only be played in a player vs. player mode. It will not provide player vs. computer functionalities, meaning no artificial intelligence will be available.

REQUIREMENTS

FUNCTIONAL REQUIREMENTS

This section explains in more detail the functional requirements. It will enumerate all requirements decided upon for this software, and will be followed for the construction of the project. The functional requirements describe an interaction between the system and its environment. Further, functional requirements describe how a system should behave given a certain stimuli. It is an abstraction of the interaction with the system, without discussing specific techniques or equipments.

Use Cases/Concrete Detailed Scenario/Functionality

3.1 - A player must be able to host a game on a LAN.

In order to host a game the user must follow the step bellow:

a. User presses the "Start Game as Server" button
b. User inputs his name
c. Enter the communication port
d. User clicks the "Wait For Client" button

Afterworth, the user must wait (listening state) for a client to connect to his server, in order to play the game.

3.2 - A player must be able to connect to a server on a LAN (For client)

In order to play a game the user must follow the step bellow:

a. User press a "Start Game as Client" button
b. User inputs his name
c. Enters the IP address of the server to which he is to be connected.
d. User enters communication port of the server.
e. User enters a nickname that he wants to use in the game
f. User clicks the Connect button
g. When the connection is establish, the user (client) is notified that the game will start in 10 seconds (Countdown)

3.3 - User can control boat direction in the Top View

To can change the direction/heading of a specific boat in two ways.

To change direction without a final destination, the user must follow the steps bellow:

a. User clicks on boat.
b. Direction and Speed of this boat appears in the context menu
c. User clicks on the Map while holding the 'D' key.

d. The boat direction is changed toward the location of the mouse click
e. The Heading of the boat is then change in the context menu

To change direction with a final destination, the user must follow the steps bellow:

a. User clicks on boat.
b. Direction and Speed of this boat appears in the context menu
c. User clicks on the Map without holding the 'D' key.
d. The boat direction is changed toward the location of the mouse click
e. The Heading of the boat is then change in the context menu

3.4 - User can control boat speed in the Top View

The user can control the boat speed in the Top View by doing these steps:

a. Selecting a boat by clicking on it
b. Pressing '+' key to increase speed or '-' key to reduce speed
c. In the context menu, the speed will change according to the key that has just been pressed

3.5 - User can aim artillery in the Firing View

The user can aim his artillery by doing the step bellow:

a. User clicks on boat from which he wants to fire
b. User presses the 'TAB' key to switch to the Firing View
c. User can move its cannon by holding the left mouse button
d. To lock (stabilized) the cannon, the user release the left mouse button

3.6 - User can fire artillery to enemy ships in the Firing View

The user can fire his artillery by doing the step bellow:

a. Use the Use Case 3.5 to aim
b. Click the 'Space Bar' to fire
c. User cannot fire until the reload time expires

3.7 - User switches from the Firing View to the Top View

If the user is in the Fire View, he can swith to the Top View by doing the following steps:

a. Press the 'TAB' key
b. The user will return the to Top View with the previously selected boat in the middle of the screen

3.8 - User switches from the Top View to the Firing View

To switch from the Top View to the Firing view, the user must follow these steps:

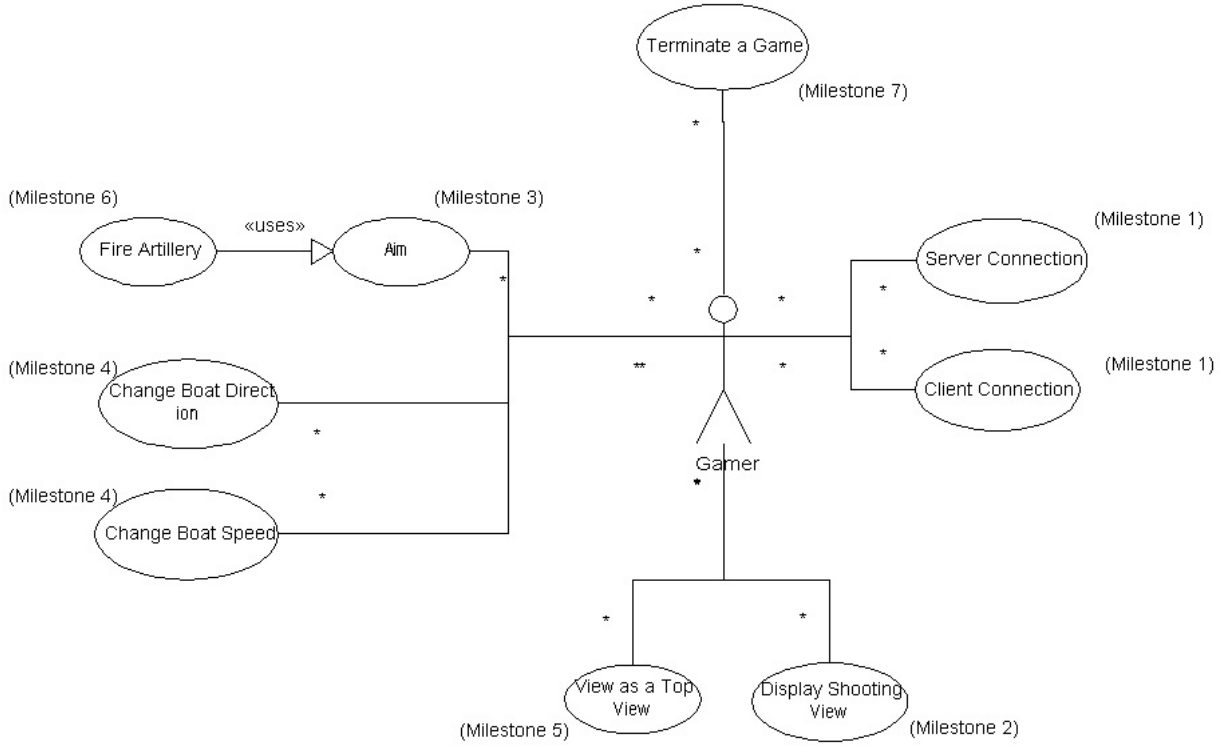
a. User selects a boat
b. User presses 'TAB' button

3.9 - User can terminate the game in both of the Views

The user can terminate the game by doing the following these steps:

a. User presses the 'ESC' key
b. A dialog appears asking the user if he wants to quit, YES or NO
c. User selects 'YES' button on this dialog

Figure 1-- Use Case Diagram



Interfaces

Session Window

The Session Window is the first window that appears when the user starts the Wave Wars application. Its three main functions are the following:

1. It must be able to start a game as client
2. It must be able to start a game as a host
3. It must let the user quit the game

The Session Window will have three buttons. The first one will be called and labeled the "Start Game as Server" button. Clicking this button will open a smaller server connection window where the user will enter the information needed to start the game as a server.

The second button called and labelled the "Start Game as Client" button. Clicking this button will open a smaller client connection window where the user will enter the information needed to start a game as a client.

The last button called and labelled the "Quit" button. Clicking this button will close the Wave Wars application.

Server Connection Window

The server connection window appears immediately after the user clicked the "Start Game as Server" button. Its main function is to acquire information needed to start a game as a server. The only information required in the port where the server will be listening and the nickname he wants to use in the game. There will be a default port 10000 written in an editable text box but the has the choice to change it.

Once the user has filled the the port and nickname text field; he is ready to start the game, he will press the "Wait For Client" button. The user can also click the "Go back to main" to return at any time to the Session Window. Once the "Wait For Client" button is pressed the "Wait Client connection" window appears. At that point the server is waiting for the opponent to connect.

If the port selected is not available, an error message will be displayed to explain that the port selected cannot be used and will suggest to change it. At that point the user has no other choice than returning to the Server Connection Window.

If there is a network problem other than a busy port, an network-error message will be displayed to user. At that point the user will have no other choice than returning to the Session Window.

Wait Client Connection Window

The "Wait Client Connection Window" appears immediately after the user has pressed "Wait For Client" in the Server Connection Window. At that point the server is listening to the specified port for a client connection. For this window a message "Waiting for a player" is displayed. This window has only one button "Cancel" which if pressed the server stops listening and goes back to the Server Connection Window.

Client Connection Window

The Client Connection Window appears immediately after the user clicked the "Start Game as Client" button as been pressed. Its main function is to acquire information needed to connect to a host in order to start a game. The information required is the server IP, the port where the server is listening and a nickname to be used in the game.

There will be three text fields to enter this information. In the port text field has a default port (port 10000) that is previously filled but user has the choice to change it.

Once the information is entered, the user will be allowed to click the "Connect" button in order to establish the connection. If a server is found the game is started on both the client side and at the server side. If the client is not able to connect to the specifed host, an error message is displayed.

Context Menu

The context menu appears as soon as a Wave Wars game starts. It is located in the bottom right corner of the game screen. The purpose of the context menu is to provide quick information on the boat that is currently selected or focused. The context menu contains a radar and combat information on the selected boat. The combat information provided is: the speed of the boat in virtual units, the boat position coordinates in the X and Y axes, the boat damage, the cannon coordinates in the X, Y and Z axes and the cannon power in virtual units. The context window does not disappears, it stays in its position no matter what the view the user is.

Top View

The Top View is the first view that appears when the game starts. In this view the user has an overview of all its boats, and possibly its opponent's boats. The purpose of the view is to give the user a good understanding of the game action.

From this view the user can also dictate the direction/speed/heading of its boat.

Firing View

The attack view can be reached by pressing the 'TAB' key. Once the tab key pressed a camera view is set behind the cannon of the focused boat. The purpose of this view is to let the change its cannon angle and fire on enemy boats. To change the angle of the cannon the user has to press the left mouse button and move the mouse. To fire on an enemy the user has to release the mouse left button, this will lock the cannon preventing it from moving it. Then the user can press the space bar to fire.

Figure 2 – Boat-by-Boat View

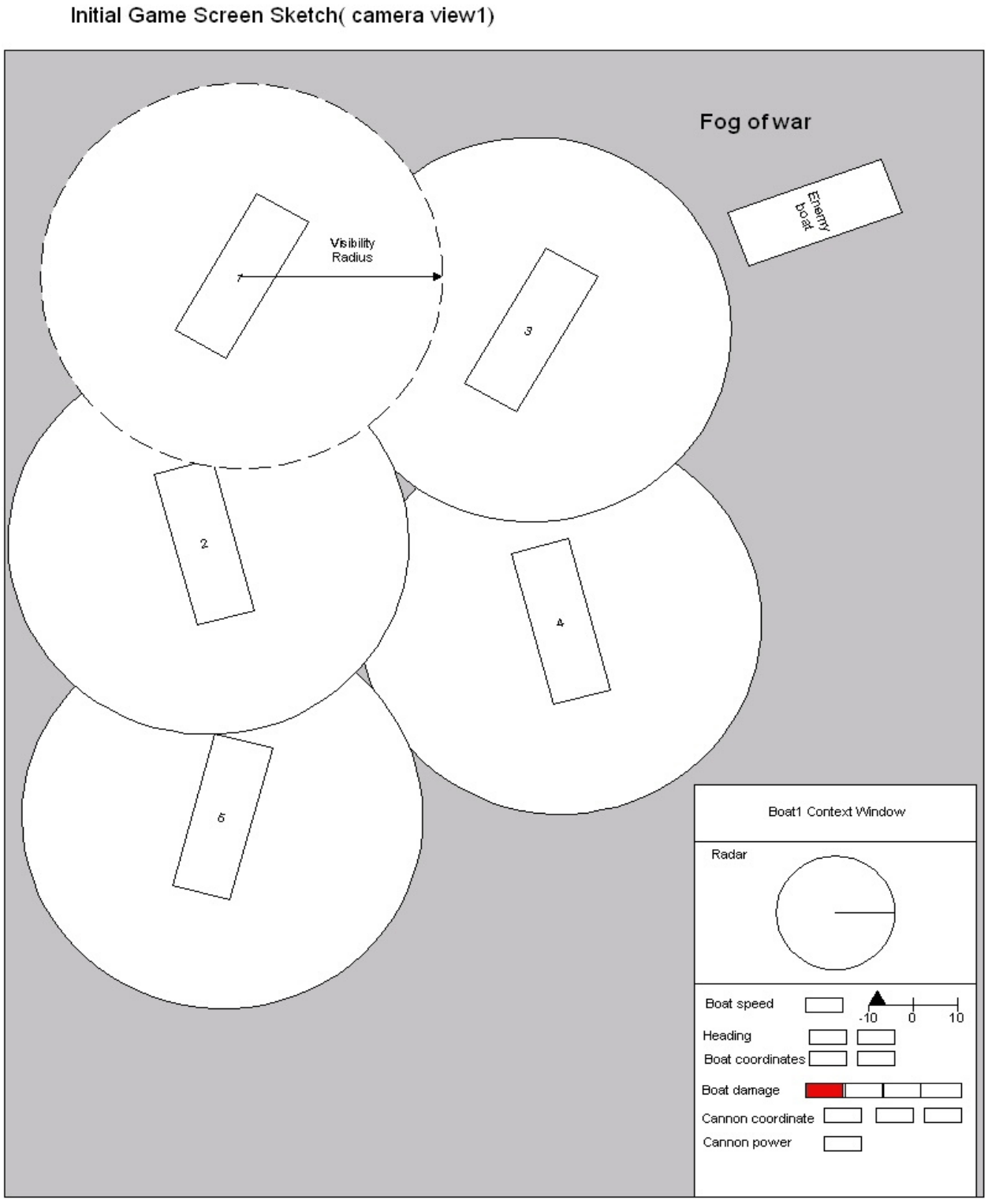
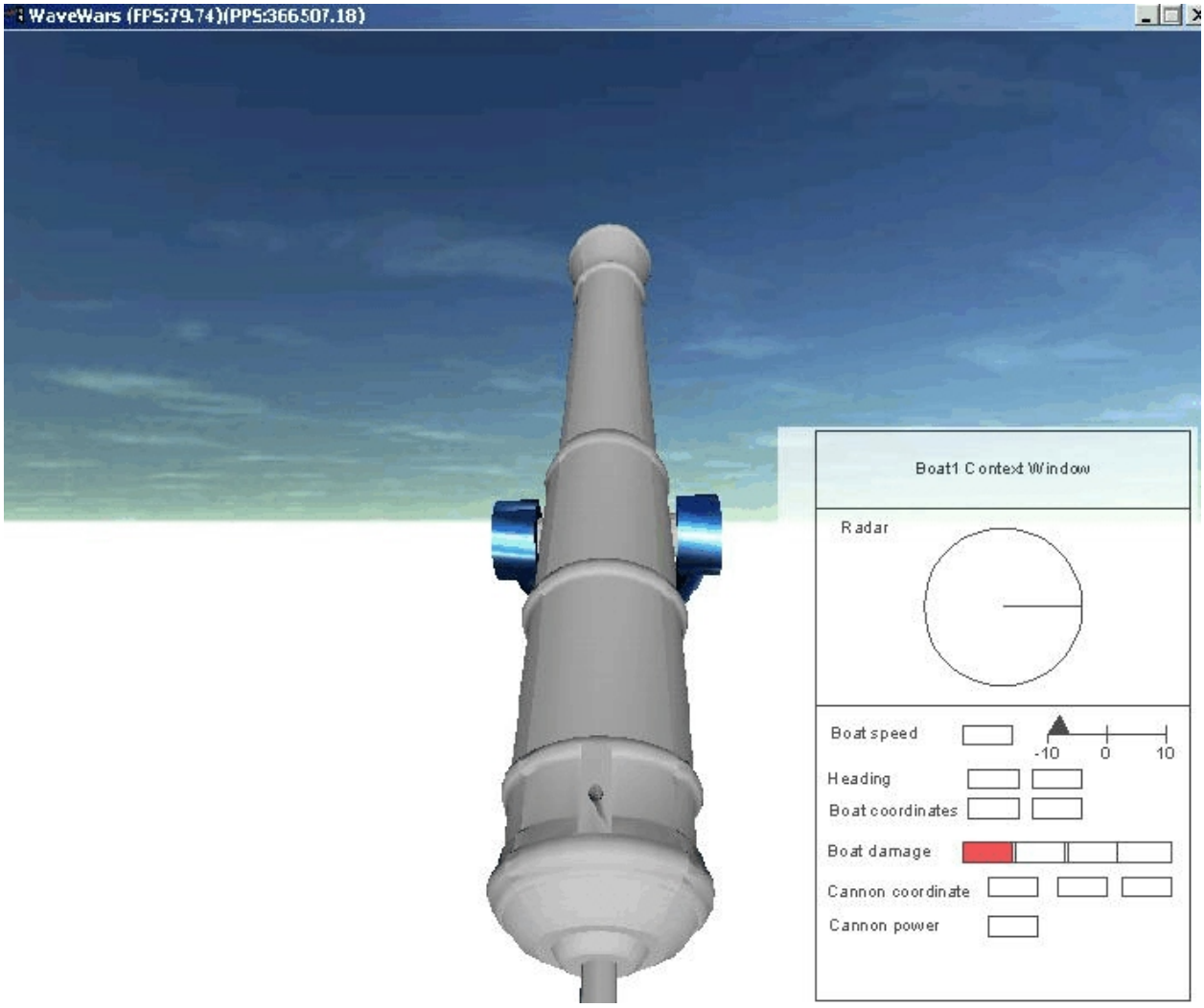


Figure 3 – Firing View



User and Human Factors

Intended audience

The project we've started is intended for the gaming industry; the user must obviously be a gamer. The user of this software will be one that is interested in a smooth and different gaming experience. Potential users also consist in people with an interest in naval warfare. However they must not expect a much high-adrenaline game such as Doom 3D where the user has to rely on his/her quick reflexes to get through the game; the user of our software will seek a game involving calculations, strategy and speed.

This piece of software should be easy to used, and should be tested by a test group formed

of programmers, the customer and some random chosen gamers before the final product is released.

Physical Characteristic

The physicals characteristics of the interaction are also important in this game, since a the public acceptance of most game is related to interaction factors such as display, controls, etc. Particular attention will be attributed to the designs of controls, physicals environment in which the action takes place, and the layout and visual qualities of the display.

Furthermore, it is also to be noted that in the first version of this game, no sound will be provided.

Controls & Display

Controls and display will be organized according to how frequently they are used, with the most commonly used controls being the most easily accessible. This requirement is in required in order to facilitate the user interaction with the system, and enhance the gaming pleasure. Furthermore, controls should also be group in relation to their functionalities, this means that related buttons should be grouped together or near one and another.

Responsiveness

The frame rate of the game will be kept high in such a way that the user cannot perceives the rate of communication, and calculation latency the game have. This is to prevent frustration from the part of the gamer. A very small response time is needed by the system to express state change to the user, this latency will be calculated by the frame rate of the system, which is set to a minimum of 25 frames per second in this game. In general, short duration and instantaneous response times are desirable in this game.

Documentation

User's documentation

User's Manual

An online (HTML) user's manual will be provided as a reference guide or tutorial for the gamers. It will begin by describing the general purpose/objective and will progress to detailed functional descriptions. This document will also describe and explain what the system do and how gamers are to do it. Furthermore, special terms, abbreviations, or acronyms used in the games will be explained in this manual.

It is also to be noted that this documentation will not be provided in the game itself. This means that won't be any special menu option to display this user manual. Brief, the user is required the read this documentation before actually playing the game.

Administrator's documentation

Customer Evaluation

This report is to be completed by the Gontran team who are working on their SEG 4000W Software Engineering Project course. ***OUR CLIENT*** will be asked by our developers team to complete this predefined form twice in the course of the year. Once when the work is nearing its half way point (April 2002) and once when the work is nearing completed (October 2002).

Design documentation

The point of this document is to communicate how we plan to build and test the system that will meet your requirements. This document captures the "blueprints" of what we will be building as well as a detailed schedule of tasks we will follow to implement this Wave War. It will be reviewed by our customer. Furthermore, our customer should approve our test report and the data we are using to validate that all are requirements have been met.

Final report

The point of this document is to communicate what we have achieved. As such it should report not only where we are at, but also how we got there and what was learned. Our customer should participate in the content of this document as well as reviewing it to ensure that they help define what was achieved and share their lessons learned as well. This report will be the historical reference for what was accomplished in your project.

Quality Assurance

The quality assurance for this software will be done in two phases,verification and validation. Both of these tests will provide us with the assurance and confidence needed to prove that the delivered system is complete and is conforming to the requirements.

General Testing Strategies

We will split the game in four main modules, which are Graphics Rendering, Network, Physics and UI . Each team member is assigned one of these modules, and will be responsible of its proper working. This means that the assigned team member is responsible of testing his own part. However, we will also have a general testing module, described in the next section as well.

Specifically, each module will be separately tested using independent black-box and gray-box methods. We will have a testing program, or driver, to try out each module to obtain a 100% statement coverage in each of the modules previously mentioned. Similarly, black-box testing will be achieved using those drivers to test the unit's functionality. Some modules will be impossible to test using this technique since it's virtually impossible to verify the validity of the output (such as a rendering engine) hence in such cases, we'll be using black-box testing to test the end cases. If all the tests pass, we will then move on to integration testing.

We are required to provide a composite class structure to be used for regression testing. This will be used every time someone will change something on their module, we will have to test the new module with every other modules again to ensure no new problems have been created by the change. By the end of regression testing, we will be able to start end-to-end testing, where we will test the system in its entirety.

Two builds of our software will co-exist. One will contain a Debug feature that will enable u to see such information as performance-monitoring performance. This build will also contain assert statements that will give us the ability to track defects that may cause unexpected behavior.

Furthermore, since the project is hosted on Sourceforge.net, we will have the ability to easily find beta testers and have many users test our game, providing further validation. Beta testing tests the system in its entirety (end-to-end again) and provides feedback from the users.

Verification

The verification of our code/architecture/design will most often be done within the life-cycle activities of our system. This will frequently take the form of formal tests, like unit testing, component testing, integration testing, regression testing, stress test, configuration test, quality test, human factor test (HCI). We will build automated test suites to handle most of this task. Each module will have its own testing module, and a more general testing module will exist to do the integration testing, as described previously.

Validation

Validation is much more subjective than verification tests, since it involves proving that within the various life-cycle activities, the customer's requirements are satisfied. This involves taking the written document and proving to the customer that each and single one of the requirements are implemented in the system. In our game, the validation will often be done by our client. We will provide him with a series of test results in the form of an html file generated by our testing program. With these files in hand, he will be able to tells us if our software meets his requirements.

Performance Testing

In our case performance testing will be used to test the responsiveness of the game, lag of the network, frame rate of the display, speed of the server, the behavior of the server under heavy stress load (volume and traffic) and the environmental tests.

Verification

The first test will be to test the main component of our system, which is the Rendering Component. This component should be tested under seriously heave load, since it's a crucial part of the system. The benchmark that we fixed for this test is that it should be capable of rendering at least 200'000 polygons/sec, plus all the normal graphical environment of the system and at the same time yielding a frame rate of >=30 frames per second. This will provide us with a good indication of

performance for this component since in this iteration of the game the maximum number of battleships will be fixed at 5 per player *(10 in total)*. This performance test will be achieved by the component test of the Graphic Rendering component. As described previously, this test will provide us with performance information output in the form of an html file.

The Network component will be the second module to be tested since it's the most used component of the game. This component should also be tested under heavy traffic load and large volumes of data transferred between the server and the client. This component will be tested using a large predefined amount of data read from a file on a client and sent through a LAN to the server. The server will process this data and send it to another client on the same LAN. At the receiver's side , the client will do a "diff" command on the received information and the expected data *(read from a file)*. Once again, this test will provide us with a status report of the network performance in the form of an html file. This document will tell us if our game satisfies our criteria of reliability *(amount of packets lost, which should be less than 2 packets per minutes)*, rapidity of the server to compute and transmit data through the LAN *(should be within a frequency higher than twice the one for the display – frame rate)*.

The third test to be executed should be a combination of both. This test will be used in conjunction with the Network component to see if we can still achieve the required frame rate with the network under such heavy load. This test will be supplied in the integration test program. It is also to be noted that this test will also provide a status output in the form of a html file, as with the other tests.

Validation

The validation technique described in the general testing strategies above will be applied here.

Functionality

Functionality is the ability of the system to do the work for which it was intended. In this game, the functionality testing will be done in order to verify and validate that all the tasks required by the system are implemented. In order to do this, all the system's components need to be working in a coordinated manner.

Verification / Validation

We will test this section by using black box testing. We will elaborate a series of test cases covering every single use case to ensure that the functionalities meet the requirements. Verification will be achieved if no error arises. Validation will be done in two phases: the first will be done by the Gontran team, the second by the customer.

HCI /Acceptance / Usability Test Strategies

These tests were designed to ensure the requirements that are dealing with the interaction of

the gamers with our system are met. Many factors have to be taken into account in this section, such as screen display, messages, ship movements, controls, responsiveness, customizability and finally the ease of use of our game.

Verification

We will run a series of tests to determine if the interface is adequate. These tests will be executed by providing sample input and evaluating the output. For example, at the beginning of the game the system will ask the user if he/she wants to start the game being the server or the client. As an input example, if the user chooses to be the client by clicking on the "Start Client Game" button, the expected output/response will be to display the client section of the wizard. If this is the response that the system exhibits during this test, then this test passes.

Validation

The cooperative evaluation is a popular way to gather information about the actual use of a system by observing the users' interaction with it. We will ask a set of randomly selected users to elaborate their actions by *'thinking aloud'*, in other words, describing what they believe is happening, why they take an action, what they are trying to do while playing with the game. We'll be asking the user questions while he's using the application *(typically of the 'why?' or 'what-if' type of questions)*. By asking these simple questions, we'll try to clarify the users' behaviour. By doing such a test, we'll be able to verify the general physical characteristics and the usability of the controls of our game.

Query techniques will also be used in this validation. Even if they are less formal than cooperative evaluation, these tests will be useful in eliciting details of the user's view of the system. Example of query technique that we will be using are questionnaires and interviews. We'll use these techniques to collect user information about the responsiveness and customizability of our game. The advantage of such methods is that they will get the user's viewpoint directly and may reveal issues which have not been considered as of yet.

The final word will be given by our client. Again, we will provide the client with a written version of the result of our tests. With this file in hand, he will be able to tells us if we are in conformance with the requirements of the game.

NON FUNCTIONAL REQUIREMENTS

The non-functional requirement or constraint describes a restriction on the system that limits our choices for constructing a solution to the problem rater than telling us what the system will do. In other word, these requirements put restrictions on the system. Theses constraints usually narrow our selection of language, platform, or implementation technique or tools.

System Requirements

As for the material needed to use the system, all that is necessary will be a PIII 1GHz PC with 128 MB of RAM, a 10/100MB network card, a Geforce 2 video card, two buttons mouse, Windows 2000, 50 MB of free hard disk space will be required for installation purposes.

Data

General Data Format

All hard-coded configuration values will be inserted into a single file which will have the macros definition for all the values we'll be using in our game.

Battleship/Missile/Artillery

For the first release of this software, we are only required to provide a single type of all object (Battleships and Weapon). An object will have a series of predefined specifications. By specifications we mean the battleship power, firing range, view range, speed and length all which will be taken from a file which will only contain macros definitions.

Input from the user will mostly be in the point-and-click format, with an occasional stroke of a key in order to specify which parameter to change. The only area where the user will actually have to enter text is when entering his name, and the server IP/Port to which he wants to connect. Otherwise, data will be input by clicking through the graphical interface.

We will create our own position measuring unit, to be used in the positioning of the boats. The data used to represent the position will be in this unit.

We will be using typical degrees for angles and the distances and power units will be virtual units only known by the system which will be constant. The speed of the boat will be set using percentages ranging from 0 to 100% speed capacity.

Resources

Programming Language

C++ will be the language used to construct this game. The software development tools *(debugger and compiler)* we will be using will be the Microsoft Visual C++ environment.

Libraries

SDL

Simple DirectMedia is a cross-platform multimedia library designed to provide fast access to the graphics frame buffer and audio device. It is used by many popular games. Simple DirectMedia Layer supports Linux, Win32, BeOS, MacOS, Solaris, IRIX, and FreeBSD. However, in this project, we will only be supporting the Windows 2000 platform.

SDL is written in C, but works with C++ natively, and has bindings to several other languages, including Ada, Eiffel, ML, Perl, PHP, Python, and Ruby. We have decided to use C++ in conjunction with SDL for this project.

The use of SDL would be justified if a second release is developed. The software could then easily be adapted to support cross-platform, since the SDL functionality would already be implemented.

OpenGL

OpenGL is the industry-standard graphics rendering library. It is deemed platform-independent and will be used for graphics rendering in our application. Its cross-platform abilities will be used, once again, in the case of a second release which would support multiple platforms. However in this release it is only used for its graphical abilities, as our software will only support the Windows 2000 platform.

Infrastructure

SourceForge (www.sourceforge.net) will permit us to exercise some configuration and version management tasks as well as bug-tracking, which are extremely important features in any software development project. The SourceForge website also lets independent users try out our software and give us feed back, a feature that will come in very handy during the testing phase of our final release. Doxygen will be used for documentation purposes.

ARCHITECTURES & DESIGN

HIGH LEVEL ARCHITECTURE

Framework/Patterns

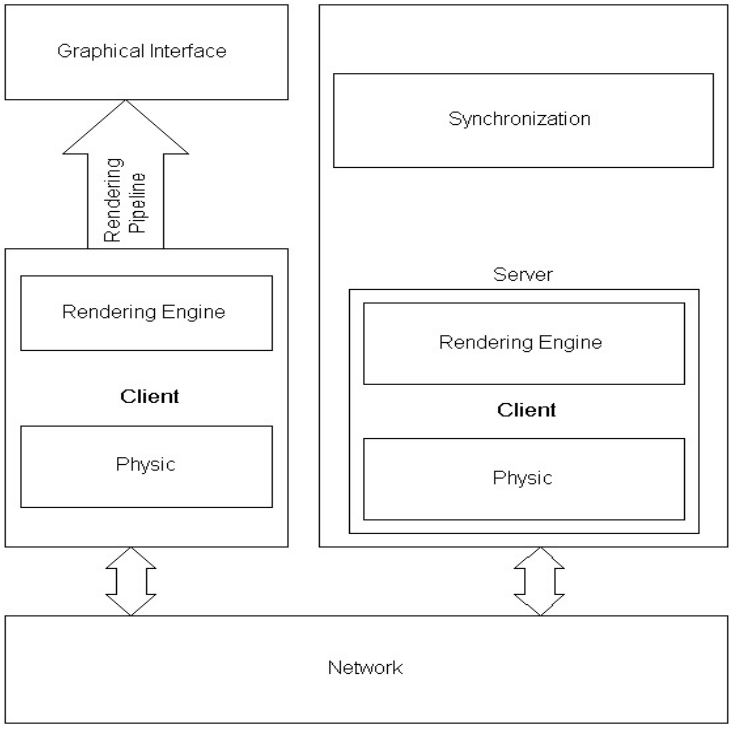
This software will be based on a client server model. However the server for a session would be the first computer who's user initiated a session; it would wait for other users to join its session. The server would be one of the players' machine.

This architecture is adequate for two reasons: first, it will help solve the network synchronization problem, since the calculations are centralized *(which is a definite advantage over the peer-to-peer architecture, which provided no easy means of solving the problem)*. Second, the calculations that will have to be executed in this game are simple enough to be done by one machine only, effectively removing the need for a distributed architecture such as peer-to-peer.

High level architecture diagram

The software architecture of this program is the structure of Wave Wars, which comprise each main components, the externally visible properties of those components, and the relationships among them. This architecture was meticulously design in order to help the development process by splitting tasks, components, ... Furthermore, the architecture was also create to minus the affect the customer requirements for the next system by presenting the customer with the opportunity to receive and upgrade system in a manner that is more reliable, timely than if the subsequent system were to be built from scratch. In conclusion, this architecture was build to lend itself to the implementation via the creation of a "skeletal" system in which the communication paths are exercised but which at first has minimal functionality.

Figure 4 – High Level Architecture

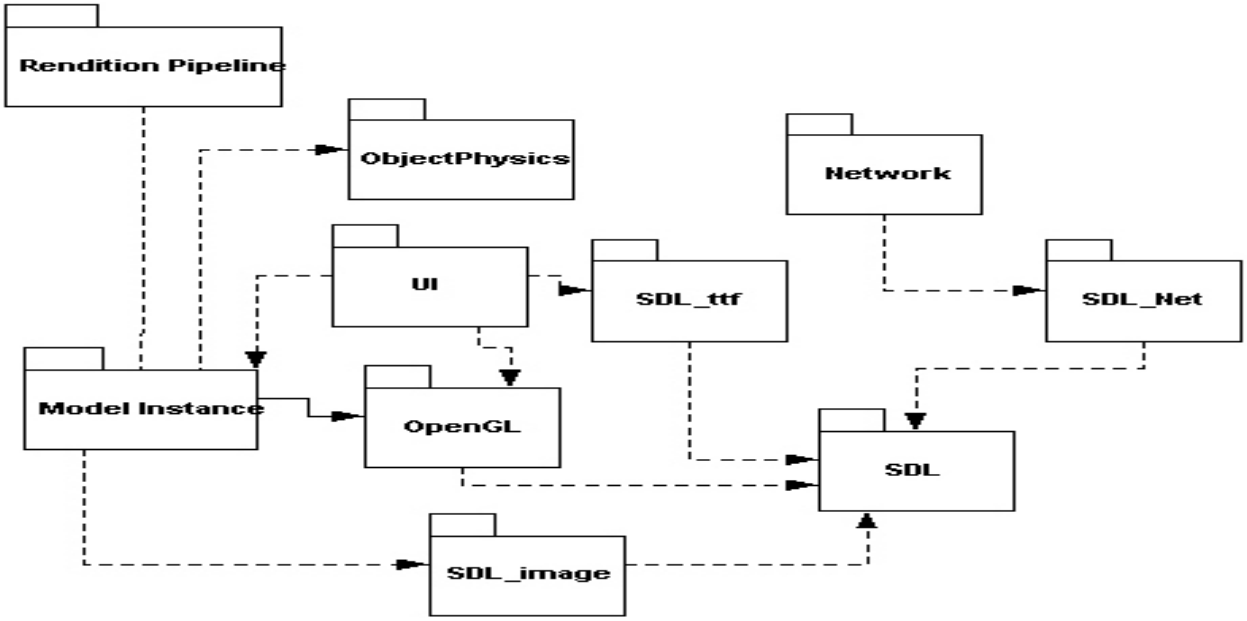


Package diagram

The package diagram below shows the packages of classes and the dependencies amongst them. A dependency exists between two elements if changes to the definition of one element may cause changes to the other. It is also to be noted that dependencies in this kind of diagram are not transitive. Finally, since dependencies imply coupling, the dependencies in the system that are shown in this diagram should be kept as minimum as possible.

To continue, this type of diagram is very useful to split class diagrams into pieces that are more easily readable than the whole class diagram itself. Another characteristic of class diagram is that it shows packages that should be used to create component tests. Basically, each package will have one or more test classes that test the behavior of the package.

Figure 5 – Package Diagram



KEY MECHANISMS AND KEY CLASSES

Classes diagram

A class diagram describes the type of objects in the system and the various kinds of static relationships that exists among them. There are two principal kinds of static relationships, which are

the associativity and subtyping of classes. Class diagrams also shows the attributes and operations of a class and the constraints that apply to the way objects are connected.

Figure 6 – Parser3DS Class Diagram

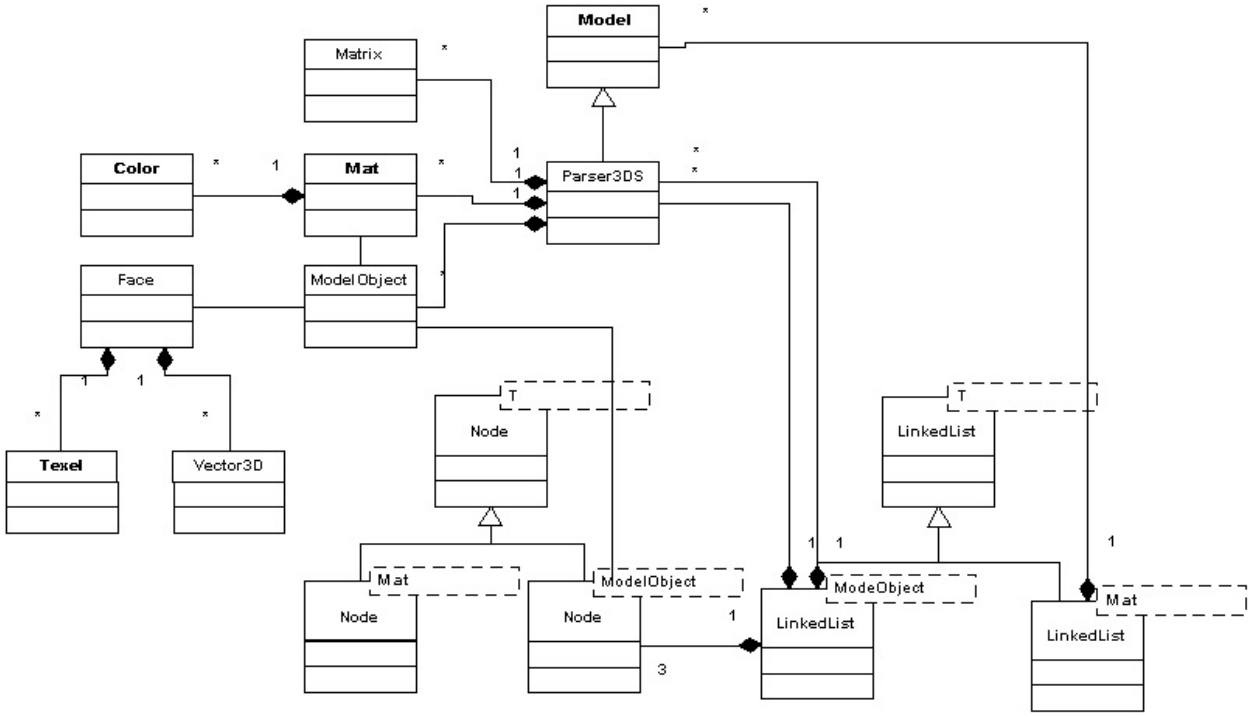
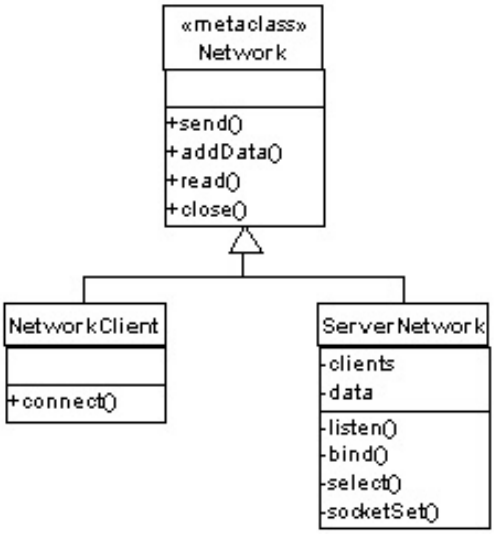


Figure 7 – Network Class Diagram



Main algorithms

Core Algorithm

1. Render the sky
2. Add water to the rendering pipeline
3. Add the visible objects to the rendering pipeline
4. Add missile and other objects to the rendering pipeline
5. Add particles to the rendering pipeline
6. Render the polygons in the pipeline
7. Render the UI (radar, number, ...)
8. Update all the objects' position according to the physic
9. Get input from the user and process the actions (include UI)
10. Send user's information to the server
11. Scan for key frame from the server (if it exists, update other players information)

Water Effect

http://www.gamedev.net/reference/articles/article915.asp
http://www.gamedev.net/reference/articles/article718.asp
http://www.gamedev.net/reference/articles/article241.asp

Collision Detection

http://www.gamedev.net/reference/articles/article974.asp
http://www.gamedev.net/reference/articles/article735.asp
http://www.gamedev.net/reference/articles/article754.asp
http://www.gamedev.net/reference/articles/article1113.asp
http://www.gamedev.net/reference/articles/article1115.asp
http://www.gamedev.net/reference/articles/article1026.asp
http://www.gamedev.net/reference/articles/article732.asp
http://www.gamedev.net/reference/articles/article1677.asp
http://www.gamedev.net/reference/articles/article736.asp
http://www.gamedev.net/reference/articles/article1234.asp
http://www.gamedev.net/reference/articles/article1057.asp
http://www.gamedev.net/reference/articles/article1112.asp
http://www.gametutorials.com/Tutorials/OpenGL/OpenGL_Pg3.htm

Networking

http://www.gamedev.net/reference/articles/article1314.asp
http://www.gamedev.net/reference/articles/article1138.asp
http://www.gamedev.net/reference/articles/article876.asp
http://www.gamedev.net/reference/articles/article701.asp

http://www.gamedev.net/reference/articles/article722.asp
http://www.gamedev.net/reference/articles/article1071.asp

Octree Partitioning Techniques

http://www.gamedev.net/reference/articles/article779.asp
http://www.gametutorials.com/Tutorials/OpenGL/OpenGL_Pg4.htm (Part 1 to 3)

Hidden Surface Removal and Ordering (HSR)

http://www.gamedev.net/reference/articles/article675.asp
http://www.gamedev.net/reference/articles/article858.asp
http://www.gamedev.net/reference/articles/article860.asp
http://www.gamedev.net/reference/articles/article805.asp
http://www.gamedev.net/reference/articles/article1088.asp
http://www.gamedev.net/reference/articles/article962.asp
http://www.gamedev.net/reference/articles/article1696.asp
http://www.gamedev.net/reference/articles/article1212.asp
http://www.gamedev.net/reference/articles/article1103.asp
http://www.gametutorials.com/Tutorials/OpenGL/OpenGL_Pg4.htm (Frustrum Culling)

Physic

http://www.gamedev.net/reference/articles/article437.asp
http://www.gamedev.net/reference/articles/article694.asp
http://www.gamedev.net/reference/articles/article1116.asp
http://www.gamedev.net/reference/articles/article1596.asp

OpenGL Tutorials

http://nehe.gamedev.net/opengl1.asp
http://nehe.gamedev.net/opengl2.asp
http://nehe.gamedev.net/opengl3.asp
http://nehe.gamedev.net/opengl4.asp
http://nehe.gamedev.net/opengl5.asp
http://nehe.gamedev.net/opengl6.asp
http://nehe.gamedev.net/opengl7.asp
http://nehe.gamedev.net/opengl8.asp

Environment Rendering

http://www.gametutorials.com/Tutorials/OpenGL/OpenGL_Pg2.htm (Sky Box)

MISCELLANEOUS

SCREEN SHOTS

Figure 8 – Dolphin Screen Shot

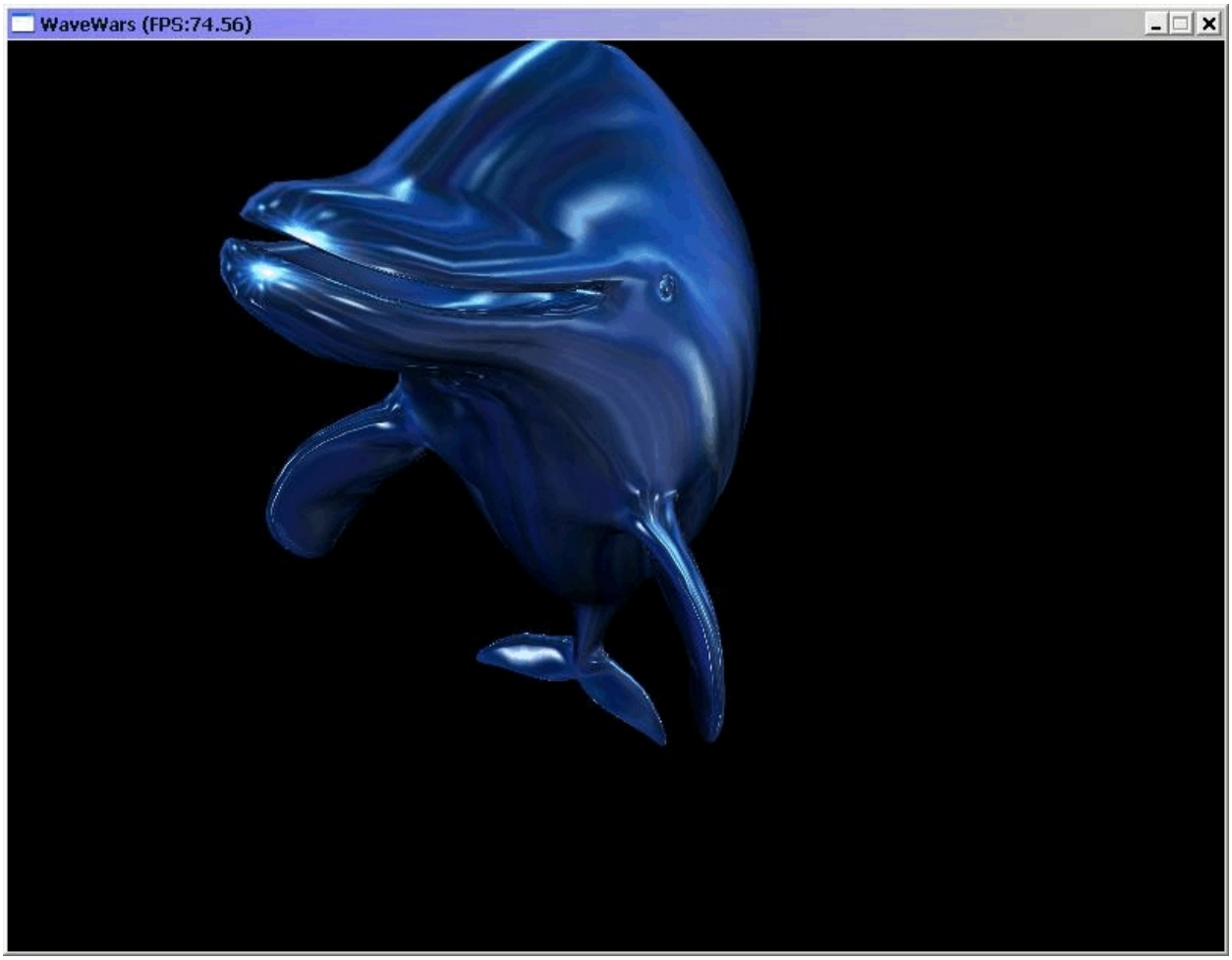
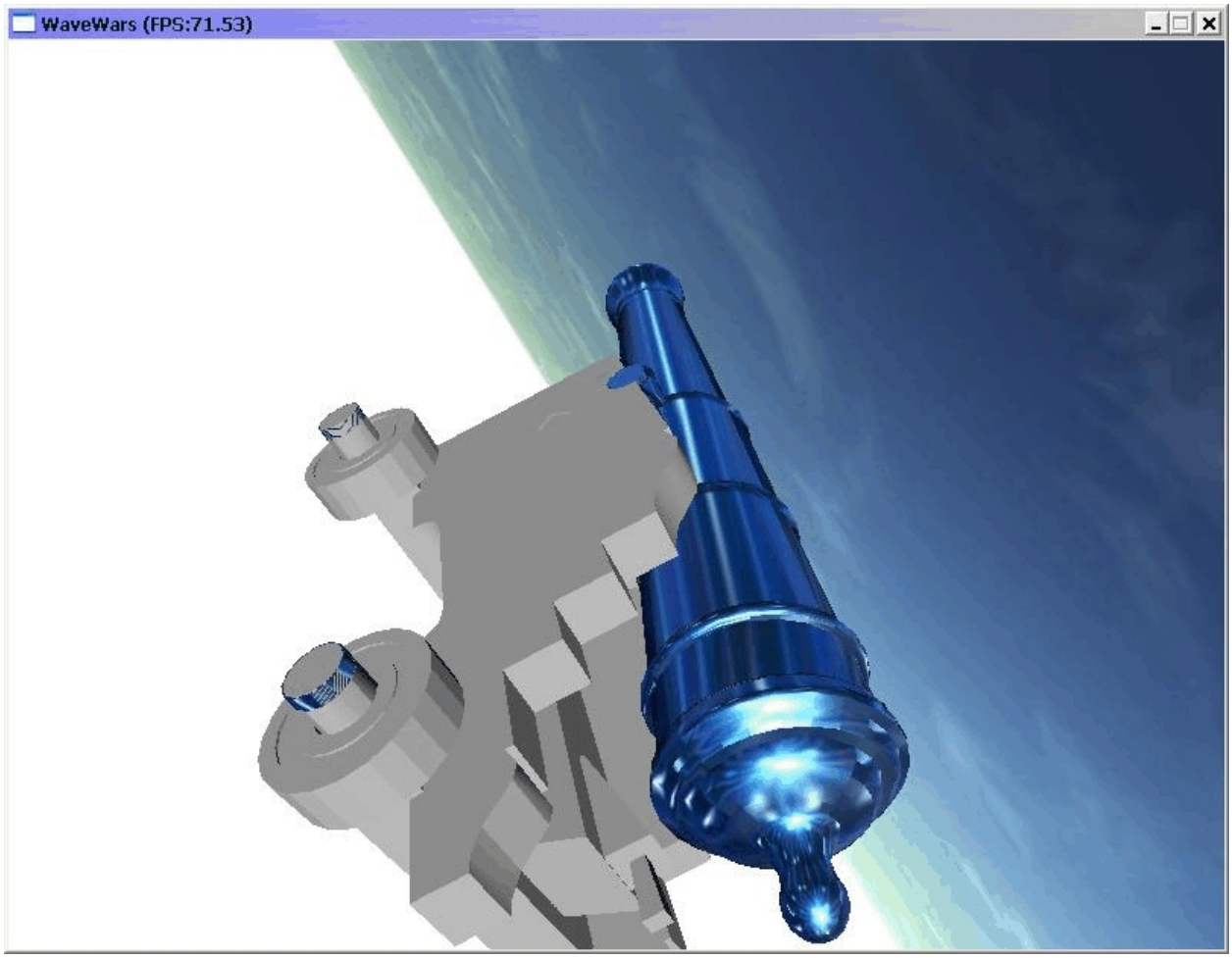


Figure 9 – Canon Screen Shot



PROJET PLAN

Our project will be sub-divided into a four releases. Each release will contain a number of new milestones. The following table summarizes our project plan.

Release Number	Milestones Incorporated	Date of Release
0.1	1) Network Handshake and Information Exchange 2) Display Firing View	May 29th, 2002
0.2	Aim Functionality 1) Change Boat Speed and Direction 2) Display Top View	June 11th, 2002
0.3	Fire Artillery and Basic Collision Detection	August 6th, 2002
1.0	Complete	August 20th, 2002

This is a draft of the project plan, and it will be revised and updated as the project evolves. We shall use Microsoft Project to manage the project plan. The first 3 releases will be incomplete versions; the last release, 1.0, is viewed as the final deliverable. Also, please note that we have provisioned for plenty of slack time; this will give us the opportunity to make changes to the project plan with enough lee-way in case we need it.

Release Number Notation

Example : Release Number 0.1.2

0.1.2
0 indicates the main release number. It is referred to as the major version number. Those releases only contain limited functionalities as indicated in the previous table, and will always work with other releases of the same major version number. For example, this release will work with 0.1.1, and any other version with a version number starting with 0. This number is incremented with each major release, which means that a major change has occurred in the software.

0.**1**.2
1 indicates the minor release number. It signifies that an important new feature has been added to the current release. However the feature does not change underlying structures and protocols, so that the minor release of a version is compatible with all other minor releases of the current version.

0.1.**2**
2 indicates the developer internal release. It signifies minor updates to the software which are not viewed as milestones or deliverables, which explains why the release numbers in the above table only contain the major and minor release number.

GLOSSARY

Word	Definitions
Adaptability	User's ability to adjust the form of input or output
Backface	The backside of a surface.
Cull	Verb (To Cull). The action of rendering what is supposed to be visible to the player, while not rendering what is not supposed to be visible. Ex. : Faces hidden by other faces will not be rendered, and so they will be culled.
Customizability	Modifiability of the user interface *(Controls, and display)* by the user or the system.
Frustrum	Set of all of the game world objects that a player can see on his screen. Equivalent to his field of vision.
HSR	Hidden Surface Removal. The use of culling.
Responsiveness	Measure of the rate of communication between the system an the user
Octree	Recursive splitting of a space into eight parts in a tree. A leaf node will be constituted of a predefined number of polygons to be accessed on demand, to be used for rendering and collision detection..
Texel	Smallest unit composing a texture. Equivalent of a pixel on a screen.

BIBLIOGRAPHY

BOOKS

Shari, Lawrence, Pfleeger. *Software Engineering – Theory and Practice*, New Jersey, 1st ed., Prentice Hall, 1998, 576p.

Alan Dix, Janet Finlay, Gregory Abowd, Russell Beale. *Human-Computer Interaction – Second Edition*, New Jersey, 2nd ed., Prentice Hall, 1998, 638p.

Len Bass, Paul Climents, Rick Kazman. *Software Architecture in Practice*, Massachusetts, 1st ed., Addison Wesley, October 1998, 452p.