Software Requirements Document

For CS372 Software Construction, Project 1

Group 5: *Cody Gaines*, *Joshua Guerrero*

Table of Contents

I. Introduction

A. Purpose

The main objective of this document is to illustrate the software requirements for the Battleship programming project. This document provides the scope and intended end goal of the project while also providing details on both the functional and nonfunctional requirements of the project. The main purpose of this project is to be able to create program that runs the game of Battleship which allows a user to compete against a computer-run opponent.

B. Scope

The project is designed to be operated by a single user and run from the command line terminal which shall work as a complete interface for the user. The game itself will follow standard Battleship rules where players take turns selecting coordinates, destroying all of the opponent's ships in order to achieve victory. The user's opponent will be the computer which shall be programmed to follow the rules of the game. At the end of each game, the user shall be able to choose to start a new game or quit out of the program.

II. Functional Requirements

A. Initialization

a. Program will begin with initialization of important entities
    a. Player objects
    b. Board objects
    c. Flags for *playerTurn* and *gameOver*
    d. Vectors for AI input validation
b. Ships are randomly placed and marked on Board objects for each player.
c. Player's board is displayed using console output into the terminal.
d. After initialization is complete, program jumps into a while loop that runs all game code.

B. Running

a. While loop is created that runs until a *gameOver* flag is detected to be true.
b. Current player is prompted to input a two-character coordinate corresponding to coordinate on board.
c. Coordinate given is saved as a *string*.
d. Program checks string for validity of coordinate
e. If valid, function call is made to change the state of the opponent's board.
f. When resolving board, check health of ship if hit. If ship is hit, check it's health. If the ship's health is zero, run *Player* member function *decShipCount* to decrease total ship count and return name of ship sunk.
g. Once board state is changed, the state of the board is outputted into the terminal for the user. The board shall be represented with ASCII characters in a 10x10 grid.
h. Once the board is displayed, the program checks the ship count of each player, starting with the opponent.
i. If a player's ship count is equal to zero, the *gameOver* flag is set to true and the while loop's conditional is fulfilled.
j. Else, the *playerTurn* flag is set to false and AI opponent is allowed to make a move.
k. At the end of AI opponent's turn, the *playerTurn* flag is set back to true and player is now allowed to make their move during next while loop iteration.
l. Code is repeated until the ship count of one player reaches zero.
m. Once *gameOver* flag is set to true, victory or defeat message is displayed and game ends.

C. Input Validation

a. In order to check validity of coordinate, the input string is passed to two functions: by itself to *check_Validity* as well as with a board object to *check_Availability*. Both functions must return *true* in order to break out of the validity check loop.
b. Input is first validated using *check_Validity* function which takes in *string* input and returns a boolean value.
c. The function first checks for the size of the input string. If the string is less than two characters, a letter and a number (e.g. A4, C3, B0, …) then a false boolean is returned. If the string passes the size validation, each character is then compared against a string containing valid coordinate characters. If the characters are allowed, then a boolean is returned as true.
d. After *check_Validity*, the input string and Board object are then passed to the function *check_Availability*, which returns a boolean value.
e. The *check_Availability* function checks the row and column of the Board object passed onto it and uses the *getHit* function to check if the section has been flagged as hit (private member boolean flag set to true). If it has, return *false*. If it has not, return *true*.
f. If either *check_Validity* or *check_Availability* return *false*, the validation loop will continue till a valid input string is received.
g. If *check_Validity* returns false, the program will prompt the player to enter a new two-character string coordinate into the console. The validation loop will then restart and check validity of the new coordinate entered into the console.
h. If *check_Validity* returns true and *check_Availability* returns false, the program will prompt the player to enter a new coordinate that has not been selected yet. The validation loop will then restart and check the validity of the new coordinate entered into the console.
i. Once both functions return *true,* the while loop will be broken and the program will be allowed to move onto board manipulation and resolution.

D. Classes Objects and Related Functions

a) Player

- All information about a player shall be stored in a *Player* class object. The code for *Player* class objects is referenced in the header file *player.h*.
- *player.h* holds constructors for the object, *get* and *set* functions, and private member variables for all *Player* objects.

b) Board

- Every coordinate is represented as a *Board* class object. Each player's board information shall be stored in a 10 by 10 two-dimensional array of type *Board*. The code for *Board* class objects is referenced in the header file *board.h*
- *board.h* holds all functions related to the board such as construction, *get* and *set* functions, and displaying board output.
- All *Board* objects shall contain a private data member that stores the name of the coordinate and flags indicating if it is occupied by a ship or if it has been a chosen coordinate yet.

E. Ships

a. Ships for each player are randomly placed on their board passing their Board object to the function *place_All_Ships.*
b. In the function *place_All_Ship*, each ship, starting from largest to smallest, is passed to the function *random_Place_Ship*.
c. The function *random_Place_Ship* takes the parameters ship length, the name of the ship, and a *Board* object.
    a. At the beginning of the function, ship orientation is randomly decided. After orientation is decided, the amount of squares the ships can be placed in is limited by the size of the ship.
    b. After ship placement is limited, a vector of all possible coordinates for placement is generated and randomized.
    c. Starting from the first coordinate in the vector, check every adjacent coordinate based on orientation. If horizontal, check every block to the left of the starting coordinate for the length of the ship. If vertical, check every block below the starting coordinate for the length of the ship.
    d. If space is occupied, mark *validSpace* boolean as false. If all spaces are unoccupied, mark them as occupied, use the member function to set the name of the spaces to the name of the ship, and set *placedShip* flag true to break out of loop.

F. Computer Opponent

a. The computer programmed opponent follows the same turn order as the user but chooses its input based on a vector of strings which contain all available coordinates.
b. If the boolean *playerTurn* is not set to true, the computer opponent is allowed to make provided an coordinate input.

c. Input is determined by passing the string vector *AITotalInput* to the function *AI_Input*. This function will return a string that will be used to mark the human player's Board object.
d. The function *AI_Input* will select a random value from the vector of total available input, copy it to a string, remove the value from the vector, and return the copied string as the chosen coordinate.
e. The computer will then wait two seconds before moving onto board manipulation and resolution.

III. Nonfunctional Requirements

A. General

a. The game is played with board consisting of a 10x10 grid.
    - Rows will be labeled A-J
    - Columns will be labeled 0-9
b. Each player is only allowed to see the placement of their own ships.
c. Each player is given a board showing which coordinates they have previously chosen and which previously chosen coordinates contain ships.
d. A player loses when all of their ships have been hit.

B. Ships

a. Each player is given a total of five ships to place on their board:

    [Carrier 5 spaces long]
    [Battleship 4 spaces long]
    [Submarine 3 spaces long]
    [Destroyer 3 spaces long]
    [Patrol Boat 2 spaces long]

b. Ships can only be placed on valid coordinates within the 10x10 board.
c. Ships are not allowed to overlap each other when placed.