Project-1

Title

Battleship Game

Course

CSC 17A

Section

48290

Due Date

30/10/2022

Author

Khadiza Akter

Contents

1. Introduction:

Battleship is a two-player strategy guessing game commonly referred to as Battleships or Sea Battle. Each player's fleet of warships is marked on regulated grids (on paper or on a board) on which the game is played. The enemy player cannot see where the ships are located. The goal of the game is to wipe out the rival player's fleet by taking turns calling "shots" at one other's ships.

The game is played on two grids, two for each player. The grids are typically square – usually 10×10 – and the individual squares in the grid are identified by letter and number. If a player successfully hit the opponent ship, then it is marked as different letter

Each participant discreetly sets their ships on their grid before the game starts. Each ship is positioned on the grid in a series of parallel locations, either horizontally or vertically. The number of positions for each ship is determined by the type of ship. The vessels cannot converge (i.e., only one ship can occupy any given position in the grid). The types and numbers of ships allowed are the same for each player. The ships should be hidden from players sight and it's not allowed to see each other's pieces. The game is a discovery game which player need to discover their opponents ship positions. The play who is able to destroy all the ships first then he/she is the winner of the game [1].

In this game, we consider 4 type of ships that are shown in Table1:

Table 1. Type of ships

Class of ships	Size
Aircraft	5
Battleship	4
Destroyer	3
Corvette	2

2. Development Summary

The project was developed using six different versions.

GameOfBattleShip_V1:

In version 1, the project was started and filled-up the 2-D matrix for player and computer, and developed the function to display the game rules.
Number of lines: 119 (including comments and spaces).

GameOfBattleShip _V2:

In version 2, the player was able to setup the ships in his/her matrix position.
Number of lines: **626 (including comments and spaces).**

GameOfBattleShip_V3:

In version 3, the computer was able to setup the ships randomly.
Number of lines: 758 (including comments and spaces).

GameOfBattleShip_V4:

In version 4, the program was able to display the play zone for both computer and player. The play can start using user input, and computer can select an attack position randomly as well.
Number of lines: 859 (including comments and spaces).

GameOfBattleShip_V5:

In version 5, check the player and computer wheather their attack was successfully or not. Update both player matrix and computer matrix for indicating miss hit as 'o' and sucessful hit as '@'. Also, in this version updated the ship status structures for computer and player and displayed status information at the top of the screen. Finally, this version determined the winner of the game.
Number of lines: 1049 (including comments and spaces).

GameOfBattleShip_Final:

In Final version, the program was able to save a game, and opened a save game successfully. The program was able to show the last game summary and last winner of the game. Also, this version was finalized the menu of the game, updating the comments, formats, lines requirement (breaking the long lines), testing the game.
Number of lines: total 1,436 (including comments and spaces);
280 lines (comments and spaces);

Lines of code: 1,156.

Number of variables: 84 (Approximate)

Number of functions: 23

I worked on the project for around three weeks and spent around 120 hours. I was not familiar with too many board games. First, I needed to understand a board game that fulfilled the project requirement and choose the battleship game. It is a two-player game, and I decided human to play with the computer. How can I able to computer play? That was a challenging part of this development. I applied several concepts from chapters 9 to 12 to complete the project and learned how to use these concepts in a software project. Apart from these chapters, I had to add concepts such as random variables, two-dimensional arrays, and current date-time functions.

3. Description

3.1 Game Rules

The game rules are summarised as follows:

1. Total four battleships for each player, the winner is who destroy other battleships first
2. The battlefield is 10x10 grid where you place all four ships
3. You can place your ships position using coordinate values (e.g., A0, B1) where 'A' or 'a' is the row and 1-10 is the column number
4. Also, you can place the ship orientation, i. e, horizontal or vertical. For horizontal orientation, type 'h' or 'H', and type 'v' or 'V' for vertical option.
5. You have total four battle ships: Aircraft Carrier-> 5 Battleship-> 4, Destroyer-> 3 and Corvette-> 2 units long.
6. You cannot place two ship at any same coordinate location.
7. After placing your ship position; you are ready to play. To attack the opponent, enter a position value such as A1 or a1, b9, j5 (without spacing) and so on.
8. If your attack is successful then it is denoted by '@', and you will continue your turn
9. If your attack is missed then it is denoted by 'o', and your turn will be end and computer will attack your ships.

3.2 How to Play the Game (Input/Output)

When run the game, it will display a menu like as Figure 1.

```
--------------------------------------------------------------------------------------------------------------
                --------------WELCOME TO BATTLESHIP GAME--------------
        Press 1= Game Rules, 2= Open a save game, 3= Last Winner, 4= Last Game Summary, X = Exit
                Any other key for start play.....
--------------------------------------------------------------------------------------------------------------
        Please choose an item...
                (1) Game Rules
                (2) Open a Saved Game
                (3) Last Winner
                (4) Last Game Summary
                (x) Exit
                ()  Any other key to play
--------------------------------------------------------------------------------------------------------------
```

Figure 1. Game Menu

If you press 1, then game rules will be displayed as Figure 2.

```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ Battleship game information/rules~ ~~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1.Total four battleships for each player, the winner is who destroy other battleships first
2.The battlefield is 10x10 grid where you place all four ships
3.You can place your ships position using coordinate values(e.g., A0, B1)where 'A' or 'a' is the row and 1-10 is the col
 number
4.Also, you can place the ship orientation, i.e horizontal or vertical. For horizontal orientation, type 'h' or 'H', and
 type 'v' or 'V' for vertical option
5.You have total four battle ships: Aircraft Carrier-> 5, Battleship-> 4, Destroyer-> 3 and Corvette-> 2 units long
6.You cannot place two ship at any same coordinate location
7.After placing your ship position; you are ready to play. To attack the opponent, enter a position value such as A1 or
a1, b9, j5 (without spacing) and so on,
8.If your attack is successful then it is denoted by '@' and you will continue your turn
9.If your attack is missed then it is denoted by 'o' and your turn will be end


-------------------------------------------------------------------------------------------------------------------
                --------------WELCOME TO BATTLESHIP GAME--------------
        Press 1= Game Rules, 2= Open a save game, 3= Last Winner, 4= Last Game Summary, X = Exit
                Any other key for start play.....
-------------------------------------------------------------------------------------------------------------------
        Please choose an item...
                (1) Game Rules
                (2) Open a Saved Game
                (3) Last Winner
                (4) Last Game Summary
                (x) Exit
                ()  Any other key to play
-------------------------------------------------------------------------------------------------------------------
```

Figure 2. Game information/rules

If you press 2, then it will open a save game (if it available). If a save game is not available then it will display the following message:

“Saved game is not available now”

If you press 3, then it will display the last winner of the game e.g., ‘human’ or ‘computer’. If it is first-game then no information available, and will display the following message:

"No information available now"

Press ‘x’ for exit the game now.

Press any other key will continue to play the game and first will ask your name, and Figure 3 shows the screen-shot of start a new game. Here player input the character ‘k’ and a new game is start and asking for player name.

```
-------------------------------------------------------------------------------------------------------------------
                --------------WELCOME TO BATTLESHIP GAME--------------
        Press 1= Game Rules, 2= Open a save game, 3= Last Winner, 4= Last Game Summary, X = Exit
                Any other key for start play.....
-------------------------------------------------------------------------------------------------------------------
        Please choose an item...
                (1) Game Rules
                (2) Open a Saved Game
                (3) Last Winner
                (4) Last Game Summary
                (x) Exit
                ()  Any other key to play
-------------------------------------------------------------------------------------------------------------------
k
Please enter your name:
Khadiza Akter
```

Figure 3. A new game start

After inputting the player name, the player will see the new screen as Figure 4 where a player setup the battle ships.

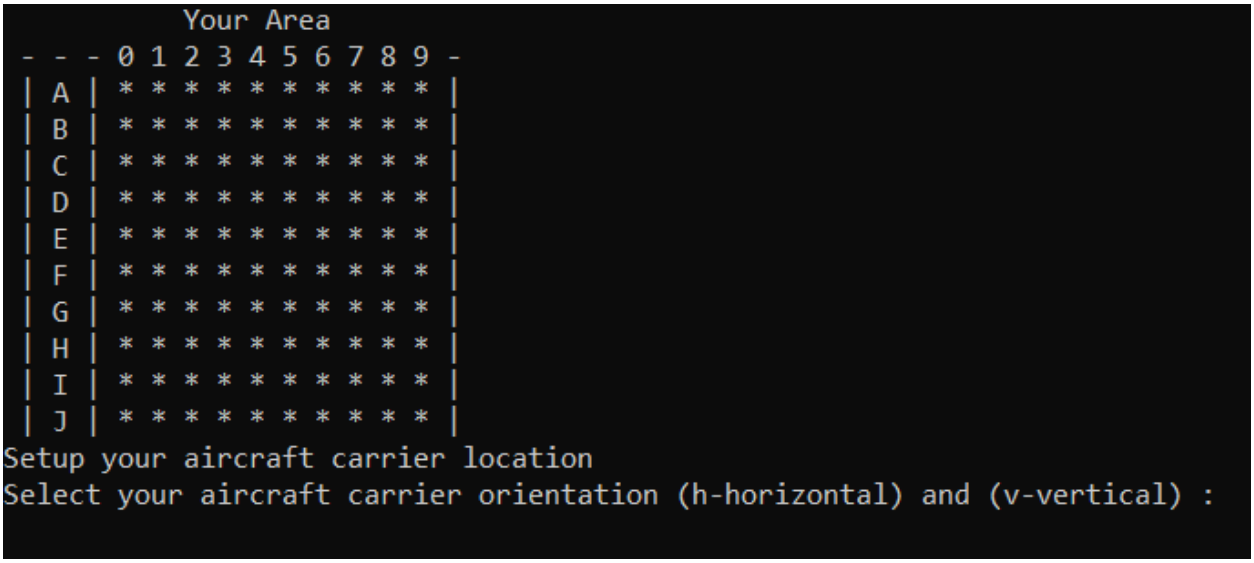


Figure 4. Setup player battle ships

At first, player will setup the aircraft location and asking for orientation of the aircraft orientation (Figure 5). The letter ‘h’ or ‘H’ will allow for horizontal orientation, and ‘v’ or ‘V’ will allow for the vertical orientation. After choosing the orientation, player will input the starting position of the aircraft location, for example, ‘a0’ or ‘A0’, start position will be the first row and column position 0. Since aircarft size is 5 units, it will be the first row and 0, 1, 2, 3, 4 columns.

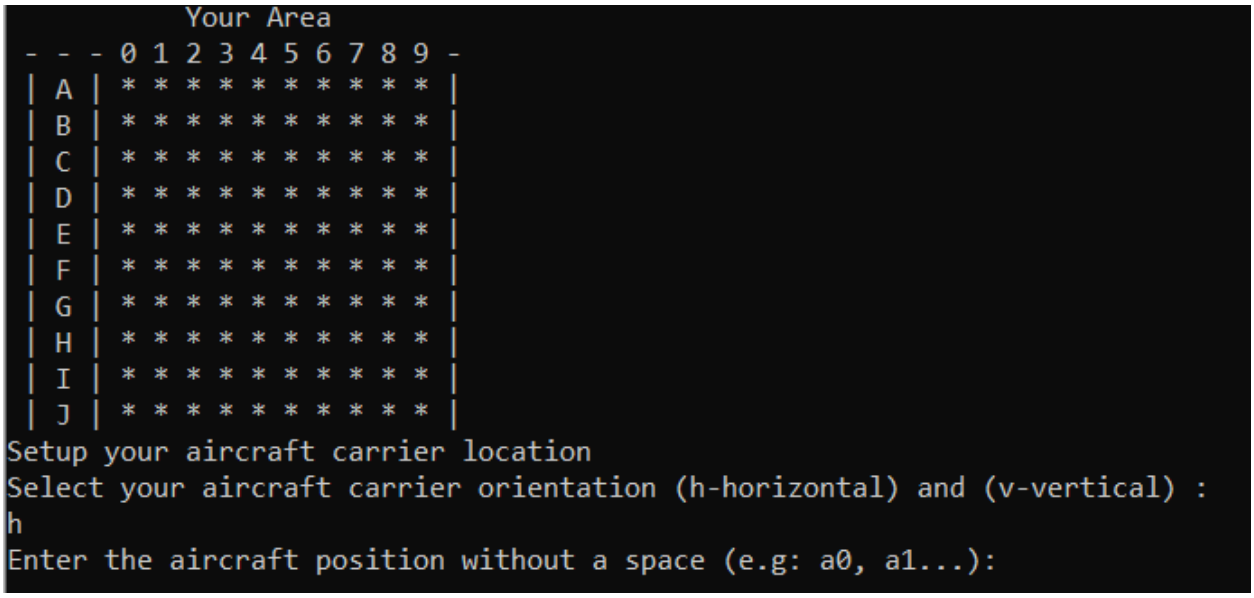


Figure 5. Aircraft setup

The Figure 6 is the screen when setup the aircraft orientation horizontal, ‘h’ and position ‘a0’, asking for setup the next ship position which is battleship.

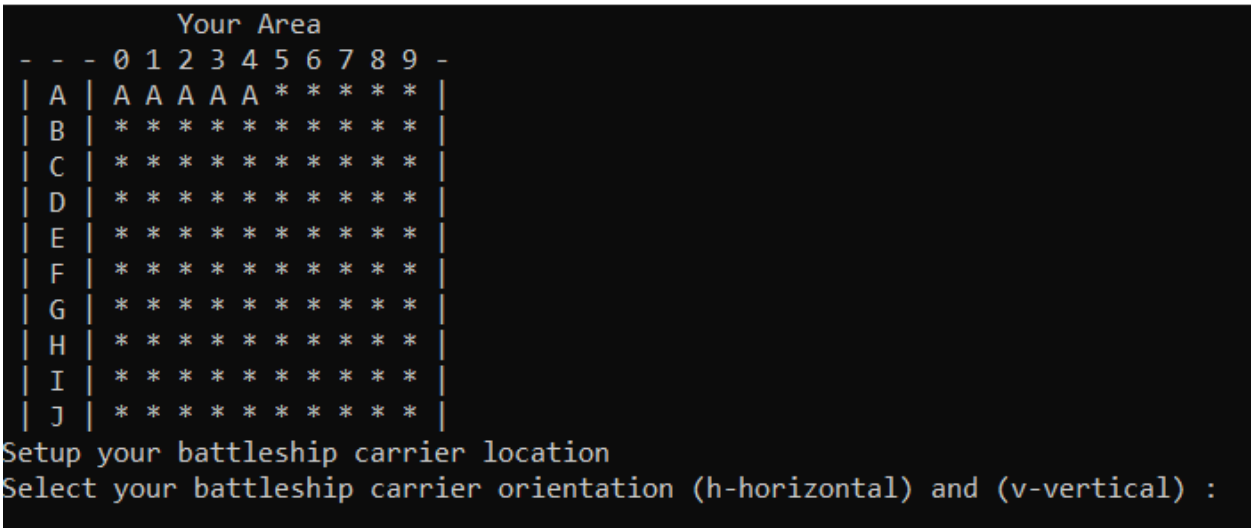


Figure 6. Setup aircraft as ‘h’ and position ‘a0’

If the player choose choose an aircraft position horizontal ‘h’ and start position a6, b6, c6 …..j6 or more then 5 units cannot be fit with four slots of the grid and return an error message to the player as Figure 7. The player can again setup the ship position.

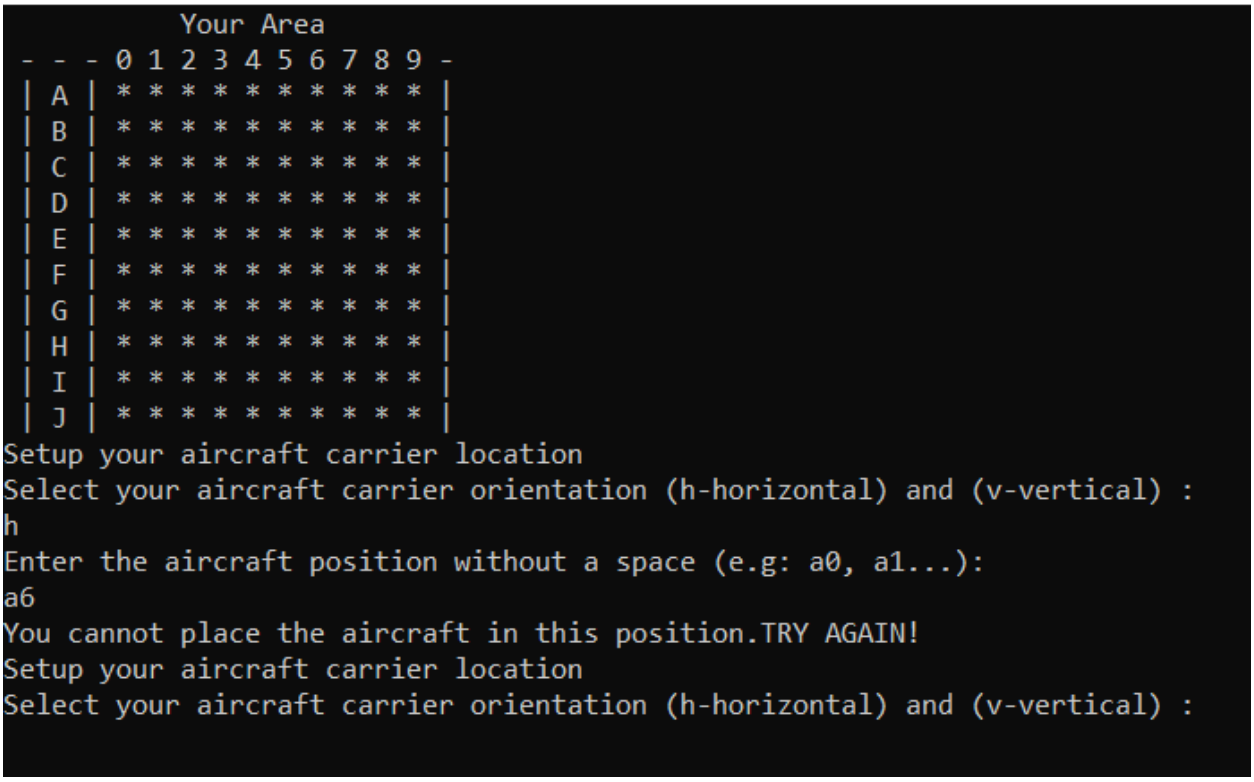


Figure 7. Setup aircraft as ‘h’ and position ‘a6’

The player will able to setup all four vehicles according to his/her choice. As soon as, the player setups all the vehicles, computer setup it’s vehicles within fraction of second, and the game playing screen will appear as Figure 8. At the top, the ship status will display for both computer and player. At the beging of play, all ships sizes are similar to the ship sizes. In the game matrix, ‘*’ represents the unexplore area, ‘o’ represents the miss hit, and ‘@’ represents the successfully hit. A player can save the game anytime by pressing the UPPER CASE ‘S’. The player will choose a position for attacking the computer’s ships. If the player chooses a wrong position, he/she will receive a message and will ask to input the position again. Figure 9 shows that the player inputs the i10 which position out of the bound and asking for to provide the input again.

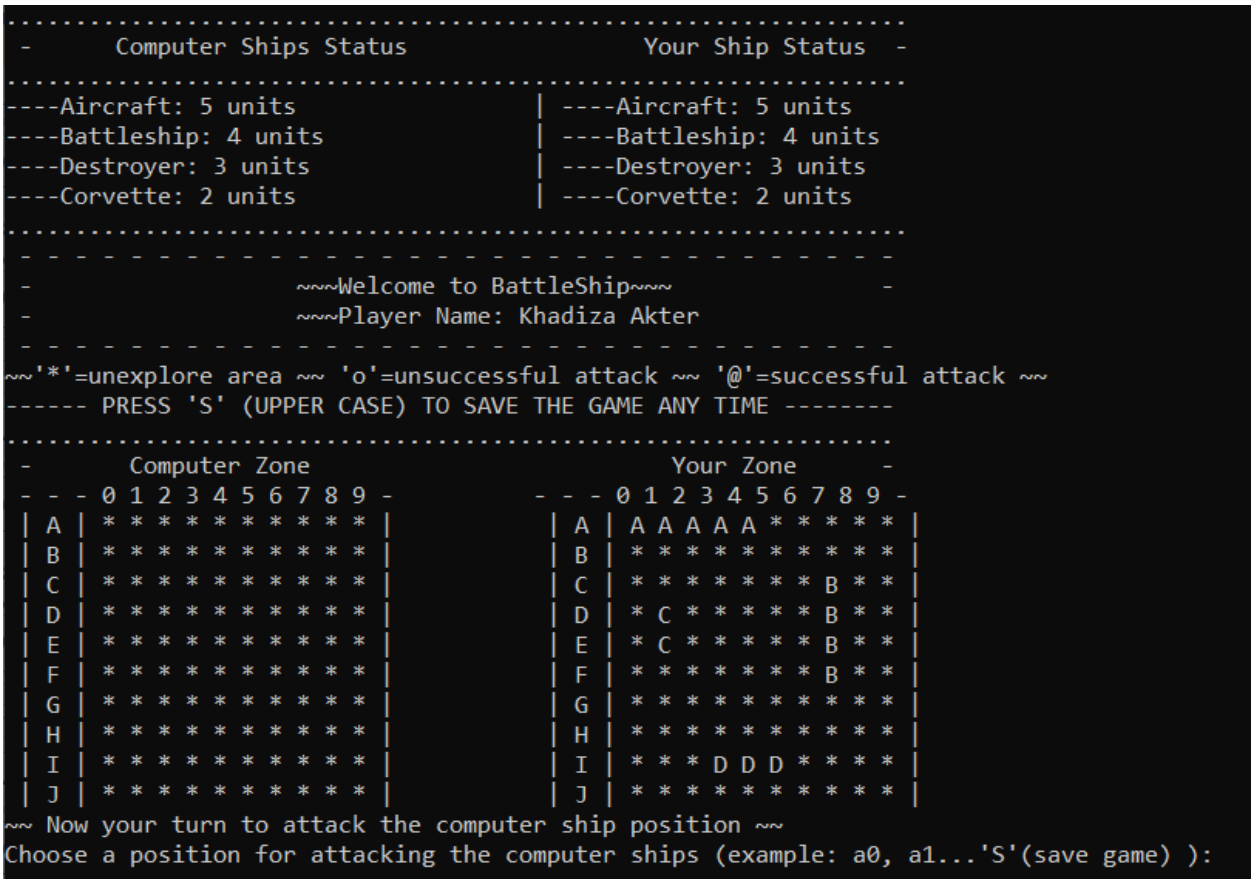


Figure 8. Game playing screen

Figure 9. Error input and asking for input position again

When the player or computer hits successfully then they allow to hit again. In the Figure 10, the player hits was successed and asked for inserting the position again.

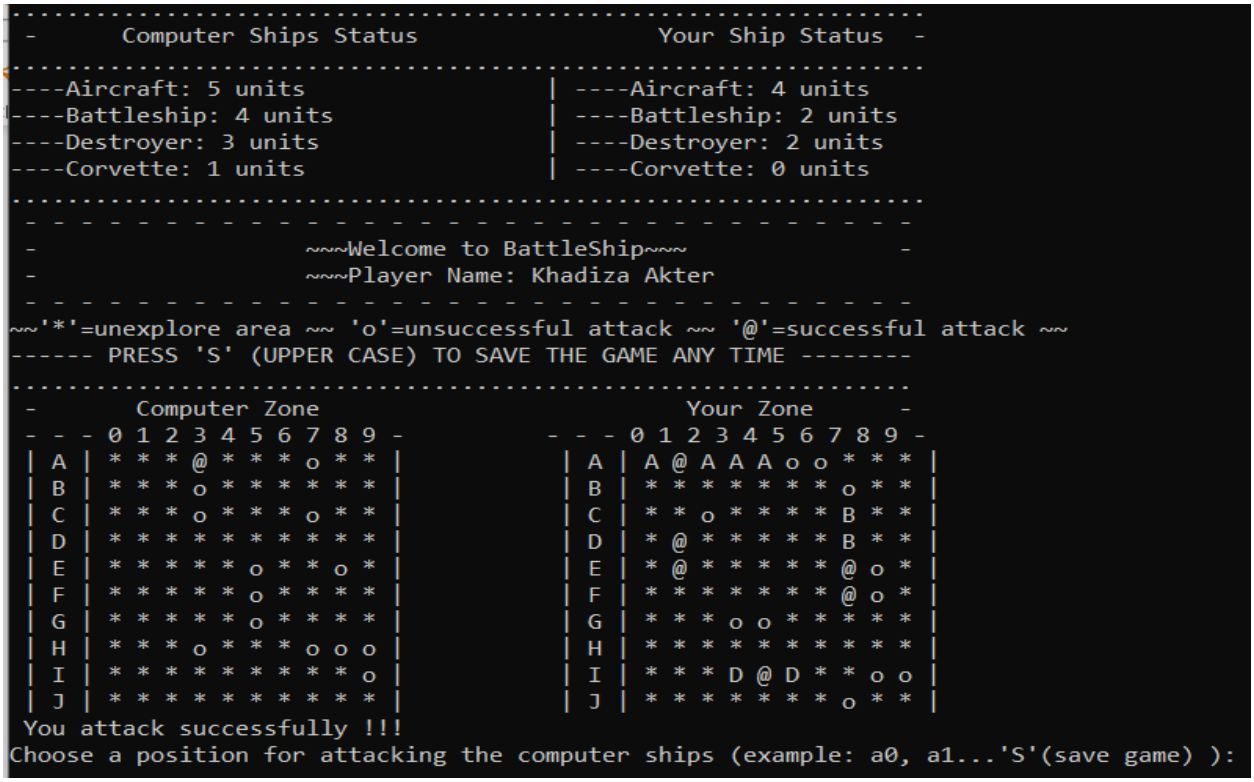


Figure 10. Player attack successful and ask for position again to hit

Also, at the top, the ship status was updated. Now **press 'S'** to save the game at this stage. When press the 'S', the game was saved and exit. Figure 11 represents the save game and exit the program. Now run the game again and choose '2' for open the save game. Figure 12 shows that the save game loaded successfully at the saving stage.

Figure 11. Save the game and exit

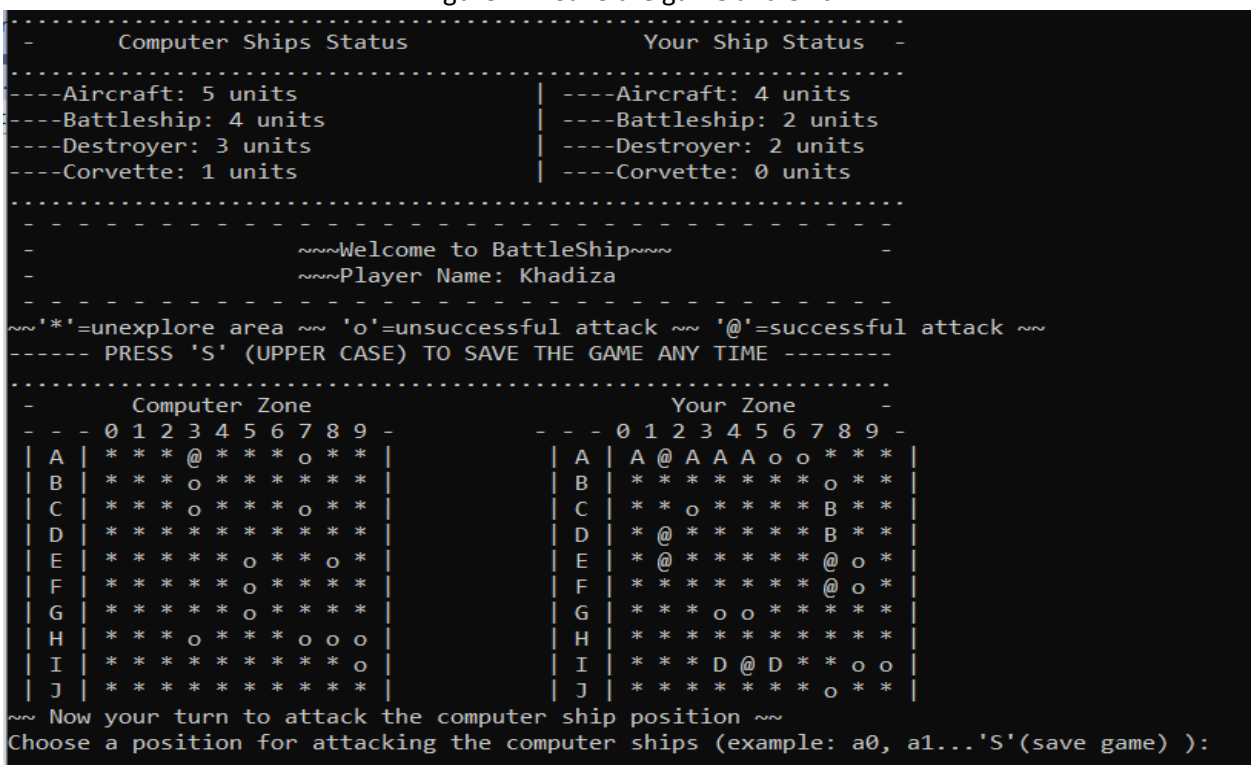


Figure 12. Open the save game

Figure 13 represents that the player won the game and exited the game.

Figure 13. Player won the game and exit

The player played a game already, and if the player run the game and choose the manu item 3 as Figure 14. It will display the last winner of the game.

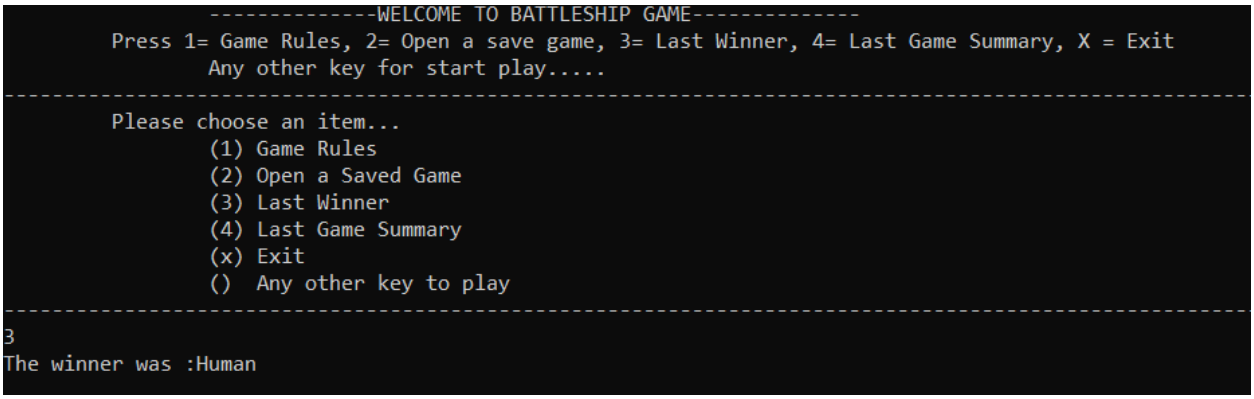


Figure 14. Last winner of the game

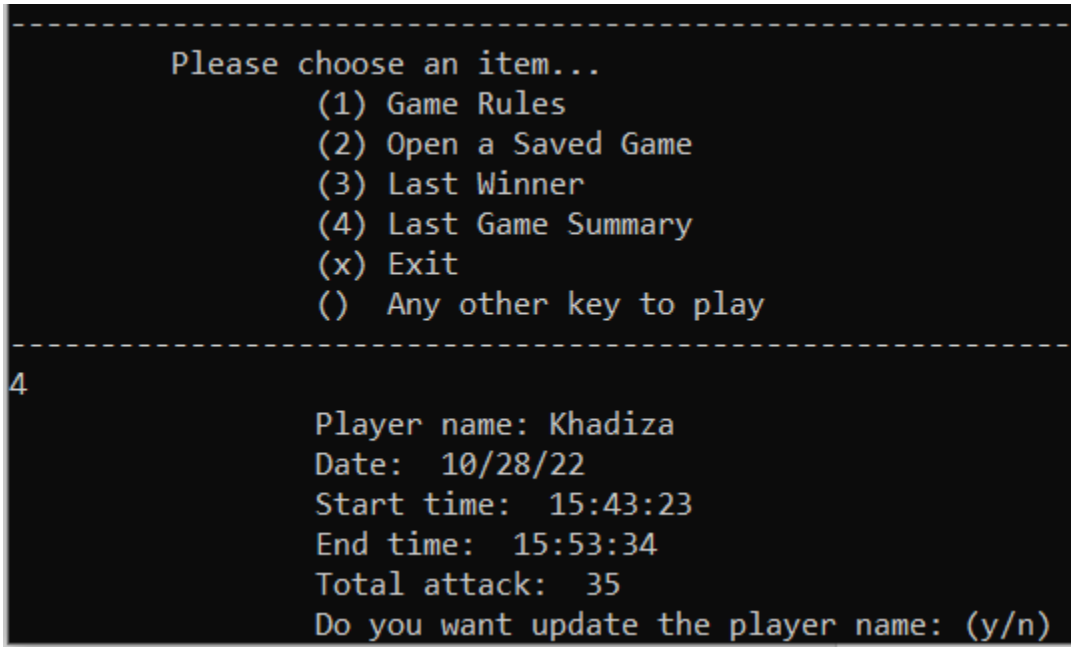


Figure 15. Shows the last game summary

Figure 15 displays the game summary. In the last game summary, the player is able to update the name, if there is any error. If the player press ‘y’, she or he can update the name as shown in Figure 16.

```
4
                Player name: Khadiza
                Date:  10/28/22
                Start time:  15:43:23
                End time:  15:53:34
                Total attack:  35
                Do you want update the player name: (y/n) y
Please input the name for updating:
Akter
```

Figure 16. Player update the name

Again, if the player wants to see the game summary then it will display the following summary as Figure 17. In this Figure, previous name Khadiza was replaced by the Akter in the game summary.

```
        Please choose an item...
                (1) Game Rules
                (2) Open a Saved Game
                (3) Last Winner
                (4) Last Game Summary
                (x) Exit
                ()  Any other key to play
------------------------------------------------------------
4
                Player name: Akter
                Date:  10/28/22
                Start time:   15:43:23
                End time:   15:53:34
                Total attack:   35
                Do you want update the player name: (y/n)
```

Figure 17. Game summary using updated name

If the player chooses the menu item 2, then it will open the saved game as Figure 10 because it was last saved game. If the player press the ‘x’, the game will be exit that shows in Figure 18.

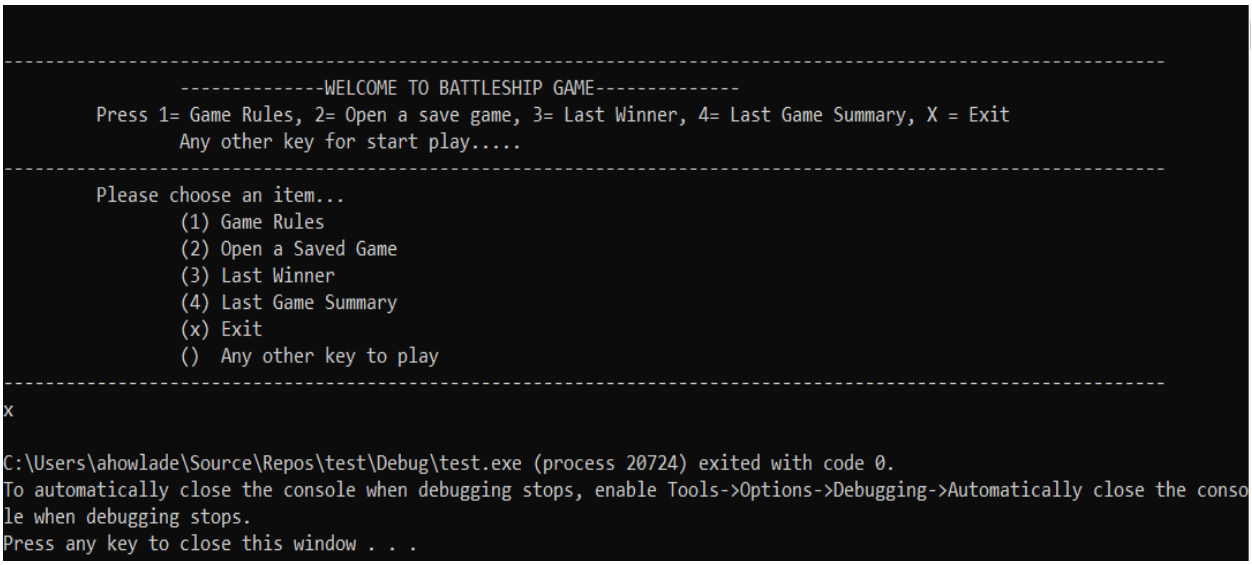
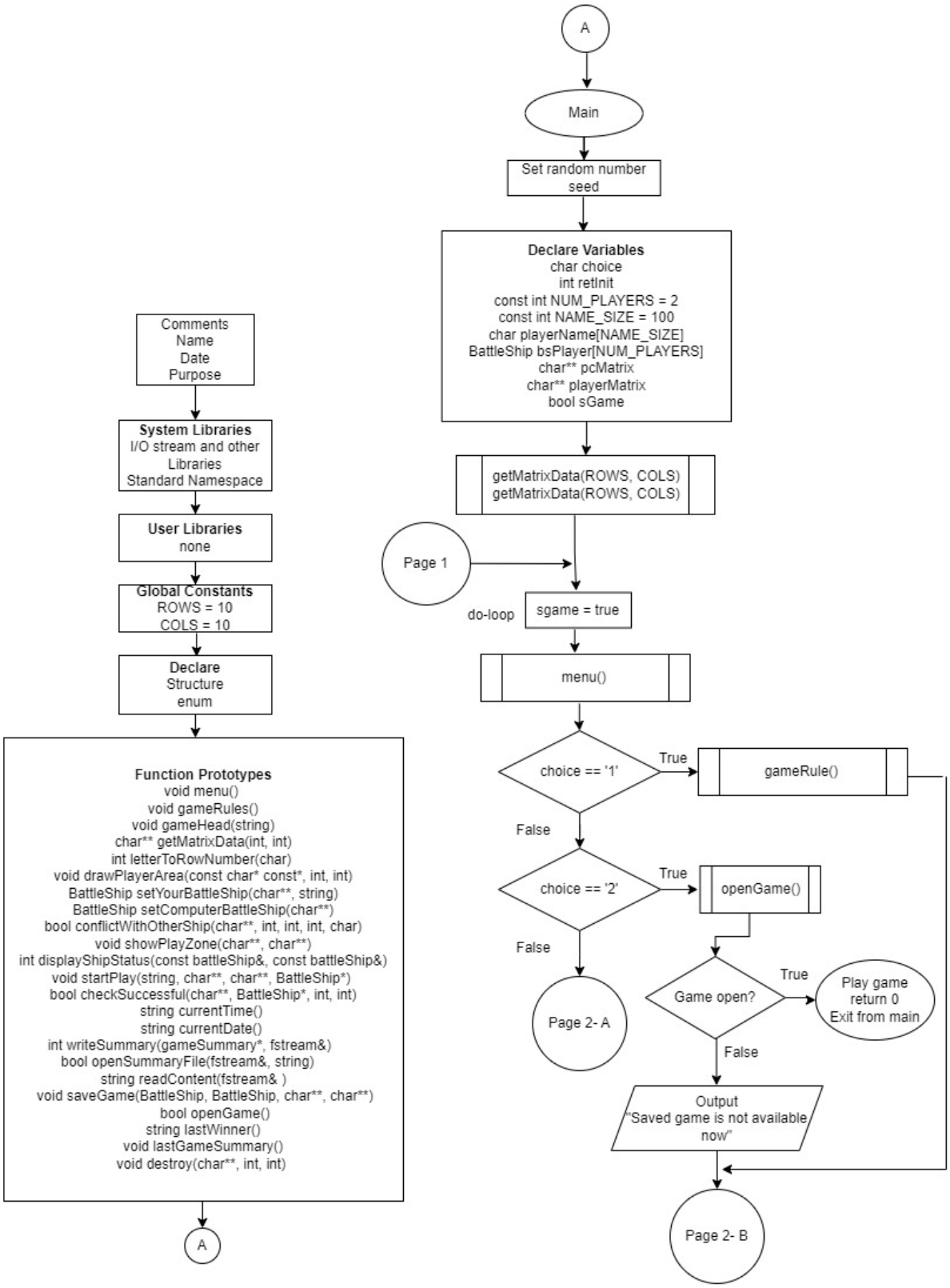


Figure 18. 'x' for exit the game

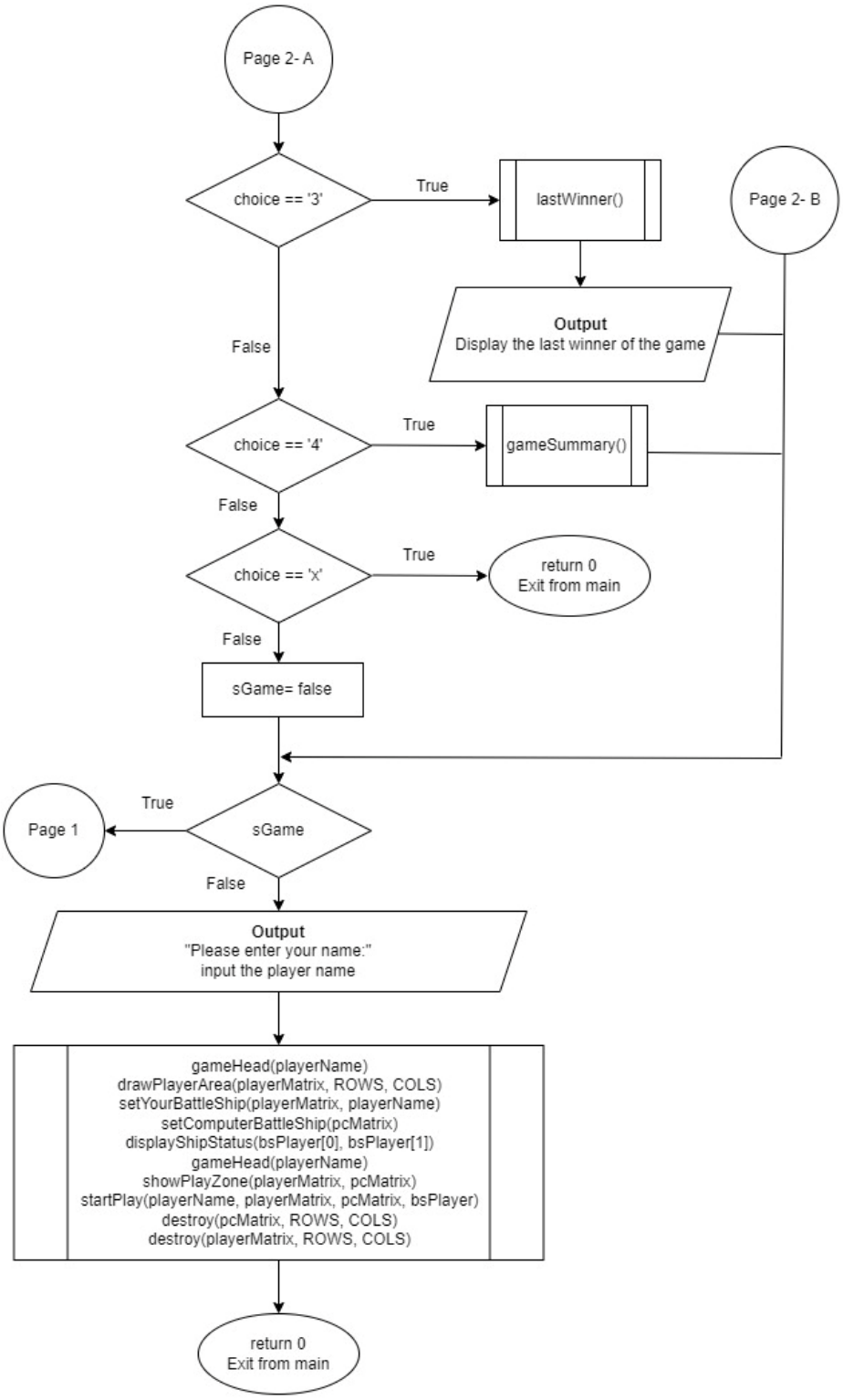
3.3 Flowchart and Psudocode of the game

3.3.1 Flowchart

The flow diagram of the game is shown in below:



Flowchart (continue)



Flowchart of the game

3.3.2 Pseudocode

The pseudocode of the program is shown in below:

```
Fill 10x10 matrix for player and computer
        repeat
            set sGame = true
            draw the menu
            choice a menu item
            if choice is 1 then
                Show the game rules
            else if choice is 2 then
                Open the save game
            else if choice is 3 then
                Display the last winner of the game
            else if choice is 4 then
                Show the last game summary
            else if choice is 'x' then
                Exit the game
            else
                Set sGame = false
            end if
        Until sGame is false
        Input the player name
        Draw the player zone only and set the battleships
        Randomly computer's set the battleships
        Display the status of ships both player and computer
        Draw the game head
        Draw the both player and computer zones
        Start the play, and save, win, or loss the game
        De-allocate the memory
        Exit the game
```

3.4 Concept Used

Chapter 9
9.2 Pointer Variables
9.5 Initializing Pointers
9.7 Pointers as Function Parameters
9.8 Dynamic Memory Allocation
9.9 Returning Pointers from Functions
Chapter 10
10.2 Character Case Conversion
10.3 C-Strings
10.4 Library Functions for Working with C-Strings
10.5 String/Numeric Conversion Functions
10.7 More about the C++ string Class
Chapter 11
11.2 Structures
11.3 Accessing Structure Members
11.5 Arrays of Structures
11.6 Focus on Software Engineering: Nested Structures
11.7 Structures as Function Arguments
11.8 Returning a Structure from a Function
11.9 Pointers to Structures
11.10 Focus on Software Engineering: When to Use., When to Use −>, and When to Use *
11.11 Enumerated Data Types
Chapter 12
12.1 File Operations
12.2 File Output Formatting
12.3 Passing File Stream Objects to Functions
12.5 Member Functions for Reading and Writing Files
12.6 Focus on Software Engineering: Working with Multiple Files
12.7 Binary Files
12.8 Creating Records with Structures
12.9 Random-Access Files
12.10 Opening a File for Both Input and Output

3.5 Major variables

Type	Name	Description	Location
Integer	ROWS	Constant rows for 10x10 matrix	global
	COLS	Constant column for 10x10 matrix	global
	NUM_PLAYERS	Number of player	main()
	NAME_SIZE	Set the maximum length of the name	main()
	AIRCRAFT_LENGTH	Unit length of the aircraft	displayShipStatus()
	BATTLESHIP_LENGTH	Unit length of the battleship	displayShipStatus()
	DESTROYER_LENGTH	Unit length of the destroyer	displayShipStatus()
	CORVETTE_LENGTH	Unit length of the corvette	displayShipStatus()
	airCarftComputer	Counter for computer air craft	displayShipStatus()
	battelShipComputer	Counter for computer battleship	displayShipStatus()
	destroyerComputer	Counter for computer destroyer	displayShipStatus()
	corvetteComputer	Counter for computer corvette	displayShipStatus()
	airCarftPlayer	Counter for player air craft	displayShipStatus()
	battelShipPlayer	Counter for player battleship	displayShipStatus()
	destroyerPlayer	Counter for player destroyer	displayShipStatus()
	corvettePlayer	Counter for player corvette	displayShipStatus()
	startPosRow	To hold the start position of rows	setYourBattleShip()
	startPosCol	To hold the start position of cols	setYourBattleShip()
	rowPosition	Randomly select a row position from 2-3	setComputerBattleShip()
	colPosition	Randomly select a column position from 2-3	setComputerBattleShip()
	shipOrientation	Select a ship orientation value 0 or 1	setComputerBattleShip()
	trackWin	The winner of the game	startPlay()
	attackCount	A counter for count the number of attack	startPlay()
char	choice	Take input for select the menu item	main()
	shipOrientiation	Take the input for ship orientation (h or v)	setYourBattleShip()
	yn	ask to get input for updating or not	lastGameSummary()
	ch	To get a char from file	lastGameSummary()
char array	playerName	Declare the input array to take player name	main()
	yCoord	declare a char array to maintain y-coordinate	showPlayZone()
	pMat	To store player matrix save data	saveGame()
	cMat	To store computer matrix save data	saveGame()

Type	Name	Description	Location
char pointer	pcMatrix	Pointer to the pc matrix	main()
	playerMatrix	Pointer to the computer	main()
	arrMatrix	Pointer to the array	getMatrixData()
string	shipPosition	Take the input for ship starting position (a0, a2...j9 so on)	setYourBattleShip()
	attackPosition	input string for attack	startPlay()
	playerName	Name of player	openGame()
	winner	To store name of winner	lastWinner()
bool	sGame	To track do while-loop for menu	main()
	isSuccessful	Successfully attack or not	startPlay()
structure	playerShipPosition	Declare a battleship structure to store player ship position	setYourBattleShip
	computerShipPosition	Declare a structure variable to store computer ship position	setComputerBattleShip()
structure array	bsPlayer	Array of structure for player and computer	main()
	bsPlayer	To read in from binary file to structure variable	openGame()
structure pointer	gSummary	Declare a structure pointer for game summary	startPlay()
enum	YCoord	Take a enum type for calculating the y-coordinate position	global

REFERENCES:

Learn the battleship game:

[1] Battleship (game), https://en.wikipedia.org/wiki/Battleship_(game)

Textbook for developing the project:

[2] Tony Gaddis, Starting with C++ from Control Structures Through Objects

In the developing phase, when I received an error or get a clearer concept regarding any issues; I searched it on the google and most of the cases I found the solution at the following references:

[3] https://stackoverflow.com/

Program Listing:

```
/*
 * File:   main.cpp
 * Author: Khadiza Akter
 * Created on October 24, 2022, 7:51 PM
 * Purpose: Simulate a BattleShip game.
          Fill 2-D pointer with '*' character
          Apply the Rules of Battleship
          Set up player battleship
          Set up computer battleship
          Draw the both matrix and playing zone
          Start the play, allow user to input and computer to select input
randomly
          Check the input of both player and computer that hit successfully or
          not, update both player matrix and ship status structure.
          Draw the number of remaining ship status both player and computer
          Winner of the game.
          Save game and game summary; open a save game and play it,
          update the main menu, project testing, update comments.
 */

 //System Level Libraries
#include <iostream> //Input-output library
#include <cstdlib>  //Srand to set the seed
#include <ctime>    //Set for time()
#include <string>   //Strings
#include <iomanip>  //Format the output
#include <fstream>  //File I/O
#include <stdlib.h> //System()

using namespace std;//Standard Name-space under which System Libraries reside

//Global Constants
const int ROWS = 10;   // Constant rows for 10x10 matrix
const int COLS = 10;   // Constant column for 10x10 matrix
//Structure declaration
struct BattleShip {    // Declare a structure for different types battle ship
    int aircraft[5][2]; //Aircraft length is 5 for tracking its coordinate value
    int battleship[4][2]; // Battleship length is 4 for tracking its coordinate
values (row,col)
    int destroyer[3][2]; // Destroyer length is 3 for tracking its coordinate
values (row,col)
    int corvette[2][2]; // Corvette length is 2 for tracking its coordinate values
(row,col)
};
struct DateTimeInfo {  // Declare a structure for date time information
    string dateOfPlay; // Playing date
    string startTimeOfPlay; // Starting time of play
    string endTimeOfPlay;   // Ending time of play
};
struct GameSummary {        // Declare a structure for game summary
    string playerName;      // Name of player
    DateTimeInfo dt;        // DateTimeInfo nested structure variable
    int totalNumberOfAttack;// Total number of attack needed for this game
    char winner;            // Who is the winner of the game
};
```

```
enum YCoord { A, B, C, D, E, F, G, H, I, J };                 // Take a enum type
for calculating the y-coordinate position

//Function Prototypes
void menu(); // Display menu
void gameRules(); //Display the game rules
void gameHead(string); //Display player name
char** getMatrixData(int, int); // Fill the computer or player matrix grid (10x10)
with '*' character
int letterToRowNumber(char); //Determine the letter(A to J) value to integer y-
axis value (0 to 9) using enum identifier
void drawPlayerArea(const char* const*, int, int); //Display player matrix
BattleShip setYourBattleShip(char**, string); //Set player ship position and
return structure
BattleShip setComputerBattleShip(char**); // Set computer battle ship randomly and
return a structure
bool conflictWithOtherShip(char**, int, int, int, char); //Check the ship position
conflict with other ship or not
void showPlayZone(char**, char**); // Draw the computer and player zone
int displayShipStatus(const BattleShip&, const BattleShip&); // Draw the ship
status and return a integer indicating the winning status
void startPlay(string, char**, char**, BattleShip*); // Allow the player and
computer for attack the ships
bool checkSuccessful(char**, BattleShip*, int, int); // Check where the attack is
successful or not
string currentTime(); // Current time of the play
string currentDate();// Current Date of the play
int writeSummary(GameSummary*, fstream&); // Write game summary
bool openSummaryFile(fstream&, string); // Open summary file
string readContent(fstream&); // Read the file content
void saveGame(BattleShip, BattleShip, char**, char**); // Save the game
bool openGame(); // Open a game
string lastWinner(); //Get name of last winner
void lastGameSummary(); // Print game summary
void destroy(char**, int, int); //De-allocate memory

//Execution begins here!
int main() {
    //random seed here
    srand(static_cast<unsigned int>(time(0)));
    //declare variables
    char choice;                    // To take input for select the menu item
    int retInit;                    // Hold a return value
    const int NUM_PLAYERS = 2;      // Number of player
    const int NAME_SIZE = 100;      // Set the maximum length of the name
    char playerName[NAME_SIZE];     // Declare the input array to take player name
    BattleShip bsPlayer[NUM_PLAYERS];// Array of structure
    char** pcMatrix;                // Pointer to the number
    char** playerMatrix;            // Point to the number
    bool sGame;                     // To track do while-loop for menu


    // Fill computer matrix grid (10x10) with '*' character
    pcMatrix = getMatrixData(ROWS, COLS);
    // Fill the player matrix grid (10x10) with '*' character
    playerMatrix = getMatrixData(ROWS, COLS);
    //Output the game statistics or menu to the screen
    do {
```

```
        sGame = true;
        menu();
        cin >> choice;  // Ask for input to see the rules or continue to game
        if (choice == '1')  gameRules(); //Call function to view the games rules
        else if (choice == '2') {
            bool ga = openGame();   // Open a save game
            if (!ga) cout << "Saved game is not available now" << endl;
            else     return 0;      // If open successfully, then play and exit
        } //End else-if
        else if (choice == '3') {
            string w = lastWinner();// Display the last time winner of the match
            cout << "The winner was :" + w << endl;
        } //End else-if
        else if (choice == '4') lastGameSummary();  // Display the game summary
        else if (choice == 'x') return 0;           // exit the program
        else                    sGame = false;      // Start to play
    } while (sGame); //End do-while loop

    cin.ignore();  //To ignore one or more characters from the input buffer
    cout << "Please enter your name:" << endl; //Ask user to enter name
    cin.getline(playerName, NAME_SIZE);       // Take the player name
    //Display player matrix
    drawPlayerArea(playerMatrix, ROWS, COLS);
    //Set player ship position and return structure
    bsPlayer[0] = setYourBattleShip(playerMatrix, playerName);
    //Set computer ship position and return structure
    bsPlayer[1] = setComputerBattleShip(pcMatrix);
    //Display the ship status and return a int value
    retInit = displayShipStatus(bsPlayer[0], bsPlayer[1]);
    //Display the player information and game head
    gameHead(playerName);
    //Display the player and computer matrix
    showPlayZone(playerMatrix, pcMatrix);
    // Start the play allow player and computer to play
    startPlay(playerName, playerMatrix, pcMatrix, bsPlayer);
    //De-allocate memory
    destroy(pcMatrix, ROWS, COLS);
    destroy(playerMatrix, ROWS, COLS);
    //Exit the program
    return 0;
} //End main function

//*************************************************************
//Definition of menu.
//Input->: None, data on menu item
//Output->:No return, This display menu item
//*************************************************************
void menu() {
    //Display menu
    cout << endl;
    cout << endl;
    cout << setfill('-') << setw(112) << "" << endl;
    cout << "\t\t --------------WELCOME TO BATTLESHIP GAME--------------\n";
    cout << "\t Press 1= Game Rules, 2= Open a save game,";
    cout << " 3= Last Winner, 4= Last Game Summary, X = Exit" << endl;
    cout << "\t\t Any other key for start play....." << endl;
    cout << setfill('-') << setw(112) << "" << endl;
    cout << "\t Please choose an item..." << endl;
```

```
    cout << "\t\t (1) Game Rules " << endl;
    cout << "\t\t (2) Open a Saved Game " << endl;
    cout << "\t\t (3) Last Winner " << endl;
    cout << "\t\t (4) Last Game Summary " << endl;
    cout << "\t\t (x) Exit " << endl;
    cout << "\t\t ()  Any other key to play " << endl;
    cout << setfill('-') << setw(112) << "" << endl;
} //End menu function

//***************************************************************
//Definition of function gameRules
//Input->: None, data on game rules
//Output->:No return, Display the game rules
//***************************************************************
void gameRules() {
    char c;
    system("cls"); //clear the screen
    cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -"
        " - - - - - - - - - - " << endl;
    cout << " - - - - -                      ~~~Welcome to BattleShip Game~~~   "
        "           - - - - - -" << endl;
    cout << "~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ Battleship game information/rules~ ~"
        "~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~" << endl;
    cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -"
        " - - - - - - - - - - -" << endl;
    cout << "1.Total four battleships for each player, the winner is who"
        " destroy other battleships first" << endl;
    cout << "2.The battlefield is 10x10 grid where you place all four ships\n";
    cout << "3.You can place your ships position using coordinate values(e.g."
        ", A0, B1)where 'A' or 'a' is the row and 1-10 is the col number\n";
    cout << "4.Also, you can place the ship orientation, i.e horizontal or "
        "vertical. For horizontal orientation, type 'h' or 'H', and type"
        " 'v' or 'V' for vertical option" << endl;
    cout << "5.You have total four battle ships: Aircraft Carrier-> 5, "
        "Battleship-> 4, Destroyer-> 3 and Corvette-> 2 units long" << endl;
    cout << "6.You cannot place two ship at any same coordinate location\n";
    cout << "7.After placing your ship position; you are ready to play. To "
        "attack the opponent, enter a position value such as A1 or a1, b9,"
        " j5 (without spacing) and so on,  " << endl;
    cout << "8.If your attack is successful then it is denoted by '@' "
        "and you will continue your turn" << endl;
    cout << "9.If your attack is missed then it is denoted by 'o'"
        " and your turn will be end" << endl;

} //End gameRules function

//***************************************************************
//Definition of function gameHead
//Input->: Use a string parameter,playerName
//Output->:No return, Display the player name
//***************************************************************
void gameHead(string playerName) {

    cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n";
    cout << " -                   ~~~Welcome to BattleShip~~~               -\n";
    cout << " -                   ~~~Player Name: " << playerName << "\n";
    cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n";
} //End gameHead function
```

```
//*****************************************************************
//Definition of function showPlayZone
//Input->: Use 2 double pointer variable as @param.
//Output->:No return, draw the computer and player play zone
//*****************************************************************
void showPlayZone(char** playerMatrix, char** pcMatrix) {
    char yCoord[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', '\0' };
//declare a char array for maintain y-coordinate
    cout << "~~'*'=unexplore area ~~ 'o'=unsuccessful attack ~~ "
        "'@'=successful attack ~~\n";
    cout <<"------ PRESS 'S' (UPPER CASE) TO SAVE THE GAME ANY TIME --------\n";
    cout <<".....................................................\n";
    cout <<" -       Computer Zone                        Your Zone      -\n";
    cout <<" - - - 0 1 2 3 4 5 6 7 8 9 -          - - - 0 1 2 3 4 5 6 7 8 9 -\n";
    for (int i = 0; i < ROWS; i++) {         //Nested loop draw the player and
computer matrix
        cout << " | " << yCoord[i] << " | "; // Set the y-column 'A' to 'J'
        for (int j = 0; j < COLS; j++) {

            if (pcMatrix[i][j] >= 'A' && pcMatrix[i][j] <= 'D') {// Hide the
computer ship position
                cout << "*" << " ";
            }
            else {
                cout << pcMatrix[i][j] << " ";   // Display the computer matrix
            } //end else
        } //end for loop
        cout << "|           | " << yCoord[i] << " | "; //set the y-column 'A' to
'J'
        for (int j = 0; j < COLS; j++)
            cout << playerMatrix[i][j] << " ";       // Display the player matrix
        cout << "|" << endl;
    } //end for loop

} //End showPlayZone function

//*****************************************************************************
//Definition of function displayShipStatus
//Input:Use 2 constant structure variable(player,computer) and pass by reference
//check the ship status using structure reference.
//Output:Return the integer, display ship status
//*****************************************************************************
int displayShipStatus(const BattleShip& player, const BattleShip& computer) {
    system("cls");                                      // Clear the screen
    //Declare variable
    const int AIRCRAFT_LENGTH = 5;              // Unit length of the aircraft
    const int BATTLESHIP_LENGTH = 4;            // Unit length of the battleship
    const int DESTROYER_LENGTH = 3;             // Unit length of the destroyer
    const int CORVETTE_LENGTH = 2;              // Unit length of the corvette
    int airCarftComputer = 0;                   // Counter for air craft
    int battelShipComputer = 0;                 // Counter for battleship
    int destroyerComputer = 0;                  // Counter for destroyer
    int corvetteComputer = 0;                   // Counter for corvette
    int airCarftPlayer = 0;                     // Counter for air craft
    int battelShipPlayer = 0;                   // Counter for battleship
    int destroyerPlayer = 0;                    // Counter for destroyer
    int corvettePlayer = 0;                     // Counter for corvette
```

```
    int *ptr  = nullptr;                        // Declare a pointer variable to
point an int
    bool isFound = false;

    //Count aircraft
    for (int i = 0; i < AIRCRAFT_LENGTH; i++) { // Check the air craft position
        if (computer.aircraft[i][0] >= 0) {     // If the position not hit yet
            airCarftComputer += 1;              // Count the computer air craft
        }
        if (player.aircraft[i][0] >= 0) {
            airCarftPlayer += 1;                // Count the player air craft
        }
    }//end for loop
    //Count battleship
    for (int i = 0; i < BATTLESHIP_LENGTH; i++) {// Check the battle ship position
        if (computer.battleship[i][0] >= 0) {    // If the position not hit yet
            battelShipComputer += 1;           // Count the computer battle ship
        }
        if (player.battleship[i][0] >= 0) {
            battelShipPlayer += 1;            // Count the player battle ship
        }
    } //end for loop

    //Count destroyer
    for (int i = 0; i < DESTROYER_LENGTH; i++) {// Check the Destroyer position
        if (computer.destroyer[i][0] >= 0) {   // If the position not hit yet
            destroyerComputer += 1;         // Count the computer destroyer ship
        }
        if (player.destroyer[i][0] >= 0) {
            destroyerPlayer += 1;            // Count the player destroyer ship
        }
    }
    ptr = &corvetteComputer;  // Store the address of corvetteComputer in ptr
pointer
    //Count corvette
    for (int i = 0; i < CORVETTE_LENGTH; i++) {   // Check the corvette position
        if (computer.corvette[i][0] >= 0) {       // If the position not hit yet
            *ptr += 1;           // Count the computer corvette ship
        }
        if (player.corvette[i][0] >= 0) {
            corvettePlayer += 1;                    // Count the player corvette
ship
        }
    }
    //Display the ship status
    cout << "..............................................................\n";
    cout << " -      Computer Ships Status                Your Ship Status  -\n";
    cout << "..............................................................\n";
    cout << "----Aircraft: " << airCarftComputer << " units                "
        "| ----Aircraft: " << airCarftPlayer << " units" << endl;
    cout << "----Battleship: " << battelShipComputer << " units               "
        "| ----Battleship: " << battelShipPlayer << " units" << endl;
    cout << "----Destroyer: " << destroyerComputer << " units                "
        "| ----Destroyer: " << destroyerPlayer << " units" << endl;
    cout << "----Corvette: " << *ptr << " units                  "
        "| ----Corvette: " << corvettePlayer << " units" << endl;
    cout << "..............................................................\n";
    if (airCarftComputer == 0 && battelShipComputer == 0 &&
```

```
        destroyerComputer == 0 && corvetteComputer == 0) { // Check all computer
ships were hit successfully
        return 0;   // Computer loss and player win
    }
    else if (airCarftPlayer == 0 && battelShipPlayer == 0 &&
        destroyerPlayer == 0 && corvettePlayer == 0) {  // Check all player ships
were hit successfully
        return 1;   // player loss and computer win
    }
    else  return 2; // Continue play

}

//*****************************************************************
//Definition of function getMatrixData                            *
//fill the computer or player matrix grid (10x10) with '*' char   *
//Return the 2-D array pointer.                                   *
//*****************************************************************
char** getMatrixData(int rows, int cols) {
    int i, j; //Loop counter variable
    char** arrMatrix; //Pointer to the array
    arrMatrix = new char* [rows]; //Allocating the row space in heap dynamically
    for (i = 0; i < rows; i++) {
        arrMatrix[i] = new char[cols];  //Allocating the column space in heap
dynamically
    }
    //read into the array
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            arrMatrix[i][j] = '*';      //Set the '*' character for the matrix
        }
    }
    return arrMatrix; //Return pointer to the array
}
//*****************************************************************
//Definition of function drawPlayerArea                           *
//Use double pointer, number of rows and cols as @param           *
//Display player matrix                                           *
//*****************************************************************
void drawPlayerArea(const char* const* matrixData, const int rows, const int cols)
{
    system("cls");
    // declare a character array for maintain y-coordinate as a character
    char yCoord[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', '\0' };
    cout << "           Your Area                          " << endl;// write a
text head for computer zone
    cout << " - - - 0 1 2 3 4 5 6 7 8 9 -           " << endl; // display the x-
coordinate value
    for (int i = 0; i < rows; i++) {  // loop to draw the 2-D matrix
        cout << " | " << yCoord[i] << " | ";
        for (int j = 0; j < cols; j++) {
            cout << matrixData[i][j] << " ";
        }
        cout << "|" << endl;
    }
} //End drawPlayerArea function
```

```
//*****************************************************************
//Definition of function setYourBattleShip                        *
//Set player ship position and return structure                    *
//*****************************************************************
BattleShip setYourBattleShip(char** playerMatrix, string playerName) {
    const int AIRC_SIZE = 5;  // Unit length of the aircraft
    const int BATTLESHIP_LENGTH = 4;// Unit length of the battleship
    const int DESTROYER_LENGTH = 3; // Unit length of the destroyer
    const int CORVETTE_LENGTH = 2;  // Unit length of the corvette
    const int POSITION_LENGTH = 2;  // Input length of a grid position
    char shipOrientiation;          // Take the input for ship orientation (h or
v)
    string shipPosition = "";       // Take the input for ship starting position
(a0, a2...j9 so on)
    bool isConflict;                // To check the ship is conflict with other
ship position
    int startPosRow;                // To hold the start position of rows
    int startPosCol;                // To hold the start position of cols

    BattleShip playerShipPosition = {}; // Declare a battleship structure to store
player ship position
    while (true) {                      // Loop for setup the aircraft position
        cout << "Setup your aircraft carrier location" << endl;
        cout << "Select your aircraft carrier orientation (h-horizontal)"
            " and (v-vertical) : " << endl;
        while (true) { // take a infinite loop to satisfy the valid input for ship
orientation
            cin >> shipOrientiation; // take the input of ship orientation 'h' or
'v'
            if (tolower(shipOrientiation) == 'h' || tolower(shipOrientiation) ==
'v') {
                cin.ignore();
                break;
            }
            else {
                cout << "Please enter a valid input 'H' or 'h' or 'V' or 'v' \n";
// if input is not h or v then ask for input again
                cin.ignore();
                continue;
            }
        }
        cout<<"Enter the aircraft position without a space (e.g: a0, a1...):\n";

        while (true) { // take a infinite loop for satisfying the valid input for
air craft position
            getline(cin, shipPosition); // get the ship position
            if (shipPosition.length() == POSITION_LENGTH) { // position length
should the 2 character length
                // make the uppercase of the input position for comparing value
and allow for lower or upper case character
                for (auto& c : shipPosition) c = toupper(c);
                if ((shipPosition[0] >= 'A' && shipPosition[0] <= 'J') &&
                    (shipPosition[1] >= '0' && shipPosition[1] <= '9')) { // check
for valid input
                    break;      // if valid input then exit the infinite while
loop
                }
                else {
```

```
                    cout << "Enter a valid aircraft position without a space "
                        "(example: a0, a1...): " << endl;   // ask for valid input
again
                    shipPosition.clear();
                    continue;
                }
            } //End if
            else {
                cout << "Enter a valid aircraft position without a space"
                    " (example: a0, a1...): " << endl;   // ask for valid input
again
                shipPosition.clear();
                continue;
            } //End else
        } //End while loop

        if (tolower(shipOrientiation) == 'h') {// check for horizontal setup
            //Get the start value of y-axis(0..9) from the letter position(A-J)
            startPosRow = letterToRowNumber(shipPosition[0]);
            startPosCol = shipPosition[1] - '0';
            if (startPosCol > AIRC_SIZE) {// not able to setup the aircraft
horizontally from this position
                cout << "You cannot place the aircraft in this position.TRY
AGAIN!\n";
                shipPosition.clear();
                continue;
            } //End if
            else {
                int counter = 0;
                for (int i = startPosCol; i < (startPosCol + AIRC_SIZE); i++) {
                    playerMatrix[startPosRow][i] = 'A';// set the player matrix
with 'A' for indicating the aircraft location
                    playerShipPosition.aircraft[counter][0] = startPosRow; //
insert the ship position values in structure variable
                    playerShipPosition.aircraft[counter][1] = i;
                    counter++; // increase the counter one
                } // End for
                break;
            } //End else
        } //End if

        if (tolower(shipOrientiation) == 'v') { // check for the vertical setup
            startPosRow = letterToRowNumber(shipPosition[0]); //Get the start
value of y-axis(0-9) from the letter position(A-J)
            startPosCol = shipPosition[1] - '0'; // make a character value to
integer

            if (startPosRow > AIRC_SIZE) { // not able to setup the aircraft
vertically from this position
                cout << "You cannot place the aircraft in this position.TRY
AGAIN!\n";
                shipPosition.clear();
                continue;
            } // End if
            else {
                int counter = 0;
                for (int i = startPosRow; i < startPosRow + AIRC_SIZE; i++) {
```

```
                    playerMatrix[i][startPosCol] = 'A'; // set the player matrix
with 'A' for indicating the aircraft location
                    playerShipPosition.aircraft[counter][0] = i; // insert the
ship position values in the structure variable
                    playerShipPosition.aircraft[counter][1] = startPosCol;
                    counter++; // increase the counter one
                } //End for
                break;
            } //End else
        } //End if
    } //End while

    drawPlayerArea(playerMatrix, ROWS, COLS);// redraw the player area with the
position of battleship
    shipOrientiation = '\0'; // reset the ship orientation
    shipPosition.clear(); // clear the shipPosition
    while (true) { // loop for setup the battleship position
        cout << "Setup your battleship carrier location" << endl;
        cout << "Select your battleship carrier orientation "
            "(h-horizontal) and (v-vertical) : " << endl;
        while (true) { // take a infinite loop for satisfying the valid input for
ship orientation
            cin >> shipOrientiation; // take the input of ship orientation 'h' or
'v'
            // compare the ship orientation input if it is 'v' or 'h' then fine
            if (tolower(shipOrientiation) == 'h' || tolower(shipOrientiation) ==
'v') {
                cin.ignore();
                break;
            }
            else {
                cout << "Please enter a valid input 'H' or 'h' or 'V' or 'v' \n";
                cin.ignore();
                continue;
            }
        } //End while loop
        cout << "Enter battleship position without a space(example: a0, a1..):\n";
        while (true) { // take a infinite loop for satisfying the valid input for
battleship position
            getline(cin, shipPosition); // get the ship position
            if (shipPosition.length() == POSITION_LENGTH) { // position length
should the 2 character length
                // make the uppercase of the input position for comparing value
and allow for lower or upper case character
                for (auto& c : shipPosition) c = toupper(c);
                if ((shipPosition[0] >= 'A' && shipPosition[0] <= 'J') &&
                    (shipPosition[1] >= '0' && shipPosition[1] <= '9')) {// check
for valid input
                    break;      // if valid input then exit the infinite while
loop
                } // End if
                else {
                    cout << "Enter a valid battleship position without a"
                        " space (example: a0, a1...): " << endl; // ask for valid
input again
                    shipPosition.clear();
                    continue;
                } // End else
```

```
            } //End if
            else {
                cout << "Enter a valid battleship position without a space "
                    "(example: a0, a1...): " << endl;   // ask for valid input
again
                shipPosition.clear();
                continue;
            } // End else

        } //End While

        if (tolower(shipOrientiation) == 'h') { // check for horizontal setup
            startPosRow = letterToRowNumber(shipPosition[0]); // Get the start
value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            startPosCol = shipPosition[1] - '0';
            isConflict = conflictWithOtherShip(playerMatrix, startPosRow,
                startPosCol, BATTLESHIP_LENGTH, tolower(shipOrientiation)); //
check the ship is conflict with other ship position

            if ((startPosCol > BATTLESHIP_LENGTH + 2) || isConflict) { // not able
to setup the battleship horizontally from this position
                cout << "You cannot place the battleship in this position. TRY
AGAIN!\n";
                shipPosition.clear();
                continue;
            }
            else {
                int counter = 0;
                for (int i = startPosCol; i < startPosCol + BATTLESHIP_LENGTH;
i++) {
                    playerMatrix[startPosRow][i] = 'B'; // set the player matrix
with 'B' for indicating the battle location
                    playerShipPosition.battleship[counter][0] = startPosRow; //
insert the ship position values in the structure variable
                    playerShipPosition.battleship[counter][1] = i;
                    counter++; // increase the counter one
                } // End for-loop
                break;
            } // End else
        } // End if

        if (tolower(shipOrientiation) == 'v') { // check for the vertical setup
            // Get the start value of y-axis(0 to 9) from the letter position(A to
J)
            startPosRow = letterToRowNumber(shipPosition[0]);
            startPosCol = shipPosition[1] - '0'; // make a character value to
integer
            isConflict = conflictWithOtherShip(playerMatrix, startPosRow,
                startPosCol, BATTLESHIP_LENGTH, tolower(shipOrientiation)); //
check the ship is conflict with other ship position

            if ((startPosRow > BATTLESHIP_LENGTH + 2) || isConflict) { // not able
to setup the battleship vertically from this position
                cout << "You cannot place the battleship in this position.TRY
AGAIN!\n";
                shipPosition.clear();
                continue;
            } // End if
```

```
            else {
                int counter = 0;
                for (int i = startPosRow; i < startPosRow + BATTLESHIP_LENGTH;
i++) {
                    playerMatrix[i][startPosCol] = 'B'; // set the player matrix
with 'B' for indicating the battle location
                    playerShipPosition.battleship[counter][0] = i; // insert the
ship position values in the structure variable
                    playerShipPosition.battleship[counter][1] = startPosCol;
                    counter++; // increase the counter one
                } // End for-loop
                break;
            } // End else

        } //End if
    } //End while-loop

    drawPlayerArea(playerMatrix, ROWS, COLS); // Redraw the player area with the
position of destroyer
    shipOrientiation = '\0'; // Reset the ship-orientation
    shipPosition.clear(); // Clear the shipPosition
    while (true) { // Loop for setup the battleship position
        cout << "Setup your destroyer carrier location" << endl;
        cout << "Select your destroyer carrier orientation (h-horizontal)"
            " and (v-vertical) : " << endl;
        while (true) { // Take a infinite loop for satisfying the valid input for
ship orientation
            cin >> shipOrientiation; // Take the input of ship orientation 'h' or
'v'
            if (tolower(shipOrientiation) == 'h' || tolower(shipOrientiation) ==
'v') {    // Compare the ship orientation input if it is 'v' or 'h' then fine
                cin.ignore();
                break;
            } // End if
            else {
                cout << "Please enter a valid input 'H' or 'h' or 'V' or 'v'\n";
// If input is not h or v then ask for input again
                cin.ignore();
                continue;
            } // End else
        } // End while-loop

        cout << "Enter the destroyer position without a space (e.g, a0, a1..):\n";
        while (true) { // take a infinite loop for satisfying the valid input for
destroyer position
            getline(cin, shipPosition); // get the ship position
            if (shipPosition.length() == POSITION_LENGTH) { // position length
should the 2 character length
                for (auto& c : shipPosition) c = toupper(c);   // make the
uppercase of the input position for comparing value and allow for lower or upper
case character
                if ((shipPosition[0] >= 'A' && shipPosition[0] <= 'J') &&
                    (shipPosition[1] >= '0' && shipPosition[1] <= '9')) { // check
for valid input
                    break;      // if valid input then exit the infinite while
loop
                } // End if
                else {
```

```
                    cout << "Enter a valid destroyer position without"
                        " a space (example: a0, a1...): " << endl;
                    shipPosition.clear();
                    continue;
                } // End else
            }
            else {
                cout << "Enter a valid destroyer position without a"
                    " space (example: a0, a1...): " << endl; // ask for valid
input again
                shipPosition.clear();
                continue;
            } // End else
        } // End while-loop

        if (tolower(shipOrientiation) == 'h') { // check for horizontal setup
            startPosRow = letterToRowNumber(shipPosition[0]);
            startPosCol = shipPosition[1] - '0';
            // check the ship is conflict with other ship position
            isConflict = conflictWithOtherShip(playerMatrix, startPosRow,
                startPosCol, DESTROYER_LENGTH, tolower(shipOrientiation));
            // Not able to setup the battleship horizontally from this position
            if ((startPosCol > DESTROYER_LENGTH + 4) || isConflict) {
                cout << "You cannot place the battleship in this position.TRY
AGAIN!\n";
                shipPosition.clear();
                continue;
            } // End if
            else {
                int counter = 0;
                for (int i = startPosCol; i < startPosCol + DESTROYER_LENGTH; i++)
{
                    playerMatrix[startPosRow][i] = 'D'; // Set the player matrix
with 'D' for indicating the battle location
                    playerShipPosition.destroyer[counter][0] = startPosRow;//
Insert the ship position values in the structure variable
                    playerShipPosition.destroyer[counter][1] = i;
                    counter++; // increase the counter one
                }
                break;
            } // End else
        } // End if

        if (tolower(shipOrientiation) == 'v') { // check for the vertical setup

            startPosRow = letterToRowNumber(shipPosition[0]);
            startPosCol = shipPosition[1] - '0'; // make a character value to
integer
            isConflict = conflictWithOtherShip(playerMatrix, startPosRow,
                startPosCol, DESTROYER_LENGTH, tolower(shipOrientiation)); //
check the ship is conflict with other ship position

            if ((startPosRow > DESTROYER_LENGTH + 4) || isConflict) { // not able
to setup the destroyer vertically from this position
                cout << "You cannot place the battleship in this position.TRY
AGAIN!" << endl;
                shipPosition.clear();
                continue;
```

```
            } // End if
            else {
                int counter = 0;
                for (int i = startPosRow; i < startPosRow + DESTROYER_LENGTH; i++)
{
                    playerMatrix[i][startPosCol] = 'D'; // set the player matrix
with 'D' for indicating the battle location
                    playerShipPosition.destroyer[counter][0] = i; // insert the
ship position values in the structure variable
                    playerShipPosition.destroyer[counter][1] = startPosCol;
                    counter++; // increase the counter one
                } // End for-loop
                break;
            } // End else

        } // End if
    } // End while-loop

    drawPlayerArea(playerMatrix, ROWS, COLS); // redraw the player area with the
position of CORVETTE
    shipOrientiation = '\0'; // reset the ship-orientation
    shipPosition.clear(); // clear the shipPosition

    while (true) { // loop for setup the battleship position
        cout << "Setup your corvette carrier location" << endl;
        cout << "Select your corvette carrier orientation "
            "(h-horizontal) and (v-vertical) : " << endl;
        while (true) { // take a infinite loop for satisfying the valid input for
ship orientation
            cin >> shipOrientiation; // take the input of ship orientation 'h' or
'v'
            // compare the ship orientation input if it is 'v' or 'h' then fine
            if (tolower(shipOrientiation) == 'h' || tolower(shipOrientiation) ==
'v') {
                cin.ignore();
                break;
            } // End if
            else {
                cout << "Please enter a valid input 'H' or 'h' or 'V' or 'v'
\n";// if input is not h or v then ask for input again
                cin.ignore();
                continue;
            } // End else
        } // End while-loop

        cout << "Enter the corvette position without a space (e.g,: a0, a1.):\n";
        while (true) {
            getline(cin, shipPosition); // get the ship position
            if (shipPosition.length() == POSITION_LENGTH) { // position length
should the 2 character length
                for (auto& c : shipPosition) c = toupper(c);   // make the
uppercase of the input position for comparing value and allow for lower or upper
case character
                if ((shipPosition[0] >= 'A' && shipPosition[0] <= 'J') &&
                    (shipPosition[1] >= '0' && shipPosition[1] <= '9')) { // check
for valid input
                    break;      // if valid input then exit the infinite while
loop
```

```
                } // End if
                else {
                    cout << "Enter a valid corvette position without a space"
                        " (example: a0, a1...): " << endl;   // ask for valid
input again
                    shipPosition.clear();
                    continue;
                } // End else
            }
            else {
                cout << "Enter a valid corvette position without a space "
                    "(example: a0, a1...): " << endl;   // ask for valid input
again
                shipPosition.clear();
                continue;
            } // end else

        } // End while-loop

        if (tolower(shipOrientiation) == 'h') { // check for horizontal setup
            startPosRow = letterToRowNumber(shipPosition[0]);      // Get the
start value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            startPosCol = shipPosition[1] - '0';
            // check the ship is conflict with other ship position
            isConflict = conflictWithOtherShip(playerMatrix, startPosRow,
                startPosCol, CORVETTE_LENGTH, tolower(shipOrientiation));

            if ((startPosCol > CORVETTE_LENGTH + 6) || isConflict) { // not able
to setup the battleship horizontally from this position
                cout << "You cannot place the corvette in this position.TRY
AGAIN!\n";
                shipPosition.clear();
                continue;
            } // End if
            else {
                int counter = 0;
                for (int i = startPosCol; i < startPosCol + CORVETTE_LENGTH; i++)
                {
                    playerMatrix[startPosRow][i] = 'C';// set the player matrix
with 'C' for indicating the battle location
                    playerShipPosition.corvette[counter][0] = startPosRow;//
insert the ship position values in the structure variable
                    playerShipPosition.corvette[counter][1] = i;
                    counter++; // increase the counter one
                }
                break;
            }
        } // End if
        if (tolower(shipOrientiation) == 'v') { // check for the vertical setup
            startPosRow = letterToRowNumber(shipPosition[0]); // Get the start
value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            startPosCol = shipPosition[1] - '0'; // make a character value to
integer
            isConflict = conflictWithOtherShip(playerMatrix, startPosRow,
                startPosCol, CORVETTE_LENGTH, tolower(shipOrientiation)); // check
the ship is conflict with other ship position
```

```
            if ((startPosRow > CORVETTE_LENGTH + 6) || isConflict) { // not able
to setup the destroyer vertically from this position
                cout << "You cannot place the battleship in this position."
                    " TRY AGAIN!" << endl;
                shipPosition.clear();
                continue;
            } // End if
            else {
                int counter = 0;
                for (int i = startPosRow; i < startPosRow + CORVETTE_LENGTH; i++)
{
                    playerMatrix[i][startPosCol] = 'C'; // set the player matrix
with 'C' for indicating the battle location
                    playerShipPosition.corvette[counter][0] = i; // insert the
ship position values in the structure variable
                    playerShipPosition.corvette[counter][1] = startPosCol;
                    counter++; // increase the counter one
                }
                break;
            } // end else
        }
    } // End while-loop
    // Redraw the player area with the position of destroyer
    drawPlayerArea(playerMatrix, ROWS, COLS);
    return playerShipPosition; // Return ship position of player structure
} // End setYourBattleShip function

//*****************************************************************
//Definition of function setComputerBattleShip
//Input->:Use double pointer as @param,Set computer ship position
//Output->:Return structure
//*****************************************************************
BattleShip setComputerBattleShip(char** computerMatrix) {
    const int AIRCRAFT_LENGTH = 5;     // Unit length of the aircraft
    const int BATTLESHIP_LENGTH = 4;   // Unit length of the battleship
    const int DESTROYER_LENGTH = 3;    // Unit length of the destroyer
    const int CORVETTE_LENGTH = 2;     // Unit length of the corvette
    // Declare a structure variable to store computer ship position
    BattleShip computerShipPosition = BattleShip();
    //setup aircraft
    int rowPosition = rand() % 2 + 2; // Randomly select a row position from 2-3
    int colPosition = rand() % 2 + 2; // Randomly select a column position from 2-
3
    int shipOrientation = rand() % 2; // Select a ship orientation value 0 or 1
    if (shipOrientation == 0) { // If value is 0 then consider the orientation as
horizontal
        int counter = 0;
        for (int i = colPosition; i < colPosition + AIRCRAFT_LENGTH; i++) {
            computerMatrix[rowPosition][i] = 'A'; // Set the computer matrix with
'A' for indicating the aircraft location
            computerShipPosition.aircraft[counter][0] = rowPosition; // Insert the
ship position values in the structure variable
            computerShipPosition.aircraft[counter][1] = i; // Insert the column
position
            counter++; // Increase the counter one
        }
    } // End if
    else { // Otherwise orientation is vertical
```

```
        int counter = 0;
        for (int i = rowPosition; i < rowPosition + AIRCRAFT_LENGTH; i++) {
            computerMatrix[i][colPosition] = 'A'; // Set the player matrix with
'A' for indicating the battle location
            computerShipPosition.aircraft[counter][0] = i; // Insert the ship
position values in the structure variable
            computerShipPosition.aircraft[counter][1] = colPosition;
            counter++; // Increase the counter one
        } //End for-loop
    } //End else

    //setup the battleship
    rowPosition = (rand() % 2) + 5; // Randomly select a row position from 5-6
    colPosition = (rand() % 2) + 5; // Randomly select a column position from 5-6
    shipOrientation = (rand() % 2); // Select a ship orientation value 0 or 1
    if (shipOrientation == 0) { // If value is 0 then consider the orientation as
horizontal;
        int counter = 0;
        for (int i = colPosition; i < colPosition + BATTLESHIP_LENGTH; i++) {
            computerMatrix[rowPosition][i] = 'B'; // Set the computer matrix with
'B' for indicating the battleship location
            computerShipPosition.battleship[counter][0] = rowPosition; // Insert
the ship position values in the structure variable
            computerShipPosition.battleship[counter][1] = i; // Insert the column
position
            counter++; // Increase the counter one
        } // End for-loop
    } // End if
    else { // Otherwise orientation is vertical
        int counter = 0;
        for (int i = rowPosition; i < rowPosition + BATTLESHIP_LENGTH; i++) {
            computerMatrix[i][colPosition] = 'B';  // Set the player matrix with
'B' for indicating the battle location
            computerShipPosition.battleship[counter][0] = i; // Insert the ship
position values in the structure variable
            computerShipPosition.battleship[counter][1] = colPosition;
            counter++; // Increase the counter one
        }
    }

    //setup the Destroyer
    rowPosition = (rand() % 2) + 8; // Randomly select a row position from 8-9
    colPosition = (rand() % 3); // Randomly select a column position from 0-2
    shipOrientation = (rand() % 2);
    if (shipOrientation == 0) { // If value is 0 then consider the orientation as
horizontal
        int counter = 0;
        for (int i = colPosition; i < colPosition + DESTROYER_LENGTH; i++) {
            computerMatrix[rowPosition][i] = 'D'; // Set the computer matrix with
'D' for indicating the battleship location
            computerShipPosition.destroyer[counter][0] = rowPosition; // Insert
the ship position values in the structure variable
            computerShipPosition.destroyer[counter][1] = i; // Insert the column
position
            counter++; // Increase the counter one
        } // end for-loop
    } // end if
```

```
    else {  // Otherwise orientation is vertical
        rowPosition = (rand() % 3); // Randomly select a row position from 0-2
        colPosition = (rand() % 2) + 8;// Randomly select a column position from
8-9
        int counter = 0;
        for (int i = rowPosition; i < rowPosition + DESTROYER_LENGTH; i++) {
            computerMatrix[i][colPosition] = 'D'; // Set the player matrix with
'D' for indicating the battle location
            computerShipPosition.destroyer[counter][0] = i; // Insert the ship
position values in the structure variable
            computerShipPosition.destroyer[counter][1] = colPosition;
            counter++; // Increase the counter one
        } //End for-loop
    } //End else

    //setup the Corvette
    shipOrientation = (rand() % 2); // Randomly select ship orientation for
destroyer
    shipOrientation = 0;
    if (shipOrientation == 0) { // If value is 0 then consider the orientation as
horizontal;
        rowPosition = (rand() % 2); // Randomly select a row position from 0-1
        colPosition = (rand() % 7); // Randomly select a column position from 0-6
        int counter = 0;
        for (int i = colPosition; i < colPosition + CORVETTE_LENGTH; i++) {
            computerMatrix[rowPosition][i] = 'C'; // Set the computer matrix with
'C' for indicating the battleship location
            computerShipPosition.corvette[counter][0] = rowPosition; // Insert the
ship position values in the structure variable
            computerShipPosition.corvette[counter][1] = i; // Insert the column
position
            counter++; // Increase the counter one
        } // End for-loop
    } // End if

    else {
        // Otherwise orientation is vertical
        rowPosition = (rand() % 5) + 2; // Randomly select a row position from 2-6
        colPosition = (rand() % 2); // Randomly select a column position from 0-1
        int counter = 0;
        for (int i = rowPosition; i < rowPosition + CORVETTE_LENGTH; i++) {
            computerMatrix[i][colPosition] = 'C'; // Set the player matrix with
'C' for indicating the battle location
            computerShipPosition.corvette[counter][0] = i; // Insert the ship
position values in the structure variable
            computerShipPosition.corvette[counter][1] = colPosition;
            counter++;// Increase the counter one
        } // End for-loop
    } //end else
    return computerShipPosition;
} //End setComputerBattleShip function

//*****************************************************************
//Definition of function letterToRowNumber.Use a char variable as @param
*
//This function will determine the letter (A, B, C...J) value to
//integer y-axis value (0,1,2...9) using enum identifier
//Return integer value
```

```
//*************************************************************************
int letterToRowNumber(char letter) {
    switch (letter) {
    case 'A':
        return A;                       // return 0 for enum identifier A
    case 'B':
        return B;                       // return 1 for enum identifier B
    case 'C':
        return C;                       // return 2 for enum identifier C
    case 'D':
        return D;                       // return 3 for enum identifier D
    case 'E':
        return E;                       // return 4 for enum identifier E
    case 'F':
        return F;                       // return 5 for enum identifier F
    case 'G':
        return G;                       // return 6 for enum identifier G
    case 'H':
        return H;                       // return 7 for enum identifier H
    case 'I':
        return I;                       // return 8 for enum identifier I
    case 'J':
        return J;                       // return 9 for enum identifier J
    } // End Switch-case
} // End letterToRowNumber function

//*************************************************************************
//Definition of function conflictWithOtherShip. Use double pointer, number
//of row, col, ship length and a char.This check the ship position
//conflict with other ship or not and return Boolean status.
//*************************************************************************
bool conflictWithOtherShip(char** playerMatrix, int row, int col, int shipLength,
char shipOrientation) {
    if (shipOrientation == 'h') { // check the ship orientation
        // loop for horizontal orientation check the column till ship length
        for (int i = col; i < col + shipLength; i++) {
            // check the character for position of the matrix, if it is not '*'
            //that means it is conflict with other ship position
            if (playerMatrix[row][i] != '*') {
                return true; // return true
            }
        } //end for-loop
    } //End if

    else {
        // for horizontal orientation check the row till ship length
        for (int i = row; i < row + shipLength; i++) {
            // check the character for position of the matrix, if it is not '*'
            //that means it is conflict with other ship position
            if (playerMatrix[i][col] != '*') {
                return true; // and return true
            }//end if
        }//end for-loop
    } // end else

    return false; // Return false
} // End conflictWithOtherShip function
```

```
//*************************************************************************
//Definition of function startPlay                    *
//Input: Player Name, Player Matrix, Computer Matrix, Battleship
//information structure as @param.This function allow to input player and
//computer attack position each other, and no return
//*************************************************************************
void startPlay(string playerName, char** playerMatrix, char** pcMatrix,
BattleShip* battleShipInfo) {
    const int POSITION_LENGTH = 2; // The input position string length always two,
for example, a0,b9, c3....
    int rowPosition;               // row value
    int colPosition;               // column value
    string attackPosition;         // input string for attack
    bool isSuccessful;             // Successfully attack or not
    int trackWin;                  // The winner of the game
    fstream gInfo;                 // Declare a fstream object
    int attackCount = 0;           // A counter for count the number of attack
    GameSummary* gSummary = nullptr; // Declare a structure pointer

    gSummary = new GameSummary;    // Allocate memory for single nested structure
    gSummary->playerName = playerName; // Hold the player name
    gSummary->dt.dateOfPlay = currentDate(); // Hold the current date using nested
structure concept
    gSummary->dt.startTimeOfPlay = currentTime(); // Hold the start time using
nested structure concept
    // declare a char array for maintain y-coordinate as a character
    char yCoord[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', '\0' };
    cout << "~~ Now your turn to attack the computer ship position ~~" << endl;
    while (true) {
        while (true) { // Take a infinite loop for satisfying the valid input for
attack position
            cout << "Choose a position for attacking the computer ships"
                " (example: a0, a1...'S'(save game) ): " << endl;
            getline(cin, attackPosition);                    // Get the ship
position
            if (attackPosition == "S") break;
            if (attackPosition.length() == POSITION_LENGTH) { // position length
should the 2 character length
                // make the uppercase of the input position for comparing value
                //and allow for lower or upper case character
                for (auto& c : attackPosition) c = toupper(c);
                if ((attackPosition[0] >= 'A' && attackPosition[0] <= 'J')
                    && (attackPosition[1] >= '0' && attackPosition[1] <= '9')) {
// check for valid input
                    // Get the start value of y-axis (0,1,2...9) from the letter
position (A,B,C....J)
                    rowPosition = letterToRowNumber(attackPosition[0]);
                    colPosition = attackPosition[1] - '0'; // Convert the column
position character to integer
                    attackCount += 1; // Count the attack
                    isSuccessful = checkSuccessful(pcMatrix, &battleShipInfo[1],
                        rowPosition, colPosition); // Check it is successfully hit
or not
                    if (isSuccessful) { // If successfully hit then player will
get another change for attack
                        trackWin = displayShipStatus(battleShipInfo[0],
                            battleShipInfo[1]); //Display the ship status of the
game, and track availability
```

```
                        gameHead(playerName); // Draw the game head
                        showPlayZone(playerMatrix, pcMatrix);// Draw the play zone
                        cout << " You attack successfully !!! " << endl;
                        attackPosition.clear();
                        if (trackWin == 0) break; // Player won the game; do not
need continue
                        continue;
                    } //end if
                    else {
                        trackWin = displayShipStatus(battleShipInfo[0],
                            battleShipInfo[1]);
                        gameHead(playerName); // Draw the game head
                        showPlayZone(playerMatrix, pcMatrix);// Draw the play zone
                        cout << "You miss the hit. Now computer's turn! \n";
                        break;// End the player attack and computer will attack
now to player battle ship
                    } //end else
                } //end if
                else {
                    cout << "--Enter a valid aircraft position without a space"
                        " (example: a0, a1...'S'(save game) )----\n";// ask for
valid input again
                    attackPosition.clear();
                    continue;
                } //end else
            }
            else {
                cout << "----Enter a valid aircraft position without a space"
                    " (example: a0, a1...'S'(save game) )----" << endl;// ask for
valid input again
                attackPosition.clear();
                continue;
            }//end else
        } //end while-loop
        if (attackPosition == "S")break;
        if (trackWin == 0) { // The player won
            cout << "Congratulation!!! ~~~" << playerName
                << "~~~ You won this game!!!" << endl;
            gSummary->winner = 'p';     // Hold player win the game
            break;
        } // end if
        while (true) { // Take a infinite loop for satisfying the valid input for
attack position
            cout << "Computer is now attacking your ships...: " << endl;
            rowPosition = rand() % 10; // randomly select a row position from 0-9
            colPosition = rand() % 10; // randomly select a column position from
0-9
            isSuccessful = checkSuccessful(playerMatrix, &battleShipInfo[0],
                rowPosition, colPosition); // Check if it is successfully hit or
not
            if (isSuccessful) {  // If successfully hit then player will get
another change for attack
                trackWin = displayShipStatus(battleShipInfo[0],
battleShipInfo[1]);
                gameHead(playerName);                   // Draw the game head
                showPlayZone(playerMatrix, pcMatrix);   // Draw the play zone
                cout << "Computer attack the position: " << yCoord[rowPosition]
                    << colPosition << endl;
```

```
                cout << "Computer attack your ship successfully !!! " << endl;
                if (trackWin == 1) break; // Computer won the game; do not need
continue
                continue;
            }//end if
            else {
                trackWin = displayShipStatus(battleShipInfo[0],
battleShipInfo[1]);
                gameHead(playerName); // Draw the game head
                showPlayZone(playerMatrix, pcMatrix); // Draw the play zone
                cout << "Computer attack the position: " << yCoord[rowPosition]
                    << colPosition << endl;
                cout << "Computer miss the hit. Now your turn! " << endl;
                break;  // End the computer attack and player will attack now to
player battle ship
            } //end else
        }//end while-loop
        if (trackWin == 1) { // The computer won
            cout << "Congratulation!!! ~~~ Computer ~~~ won this game!!!\n";
            gSummary->winner = 'c';                 // Hold the winner
            break;
        }//end if
    } //End while-loop
    gSummary->totalNumberOfAttack = attackCount;    //Hold the number of attack
count
    gSummary->dt.endTimeOfPlay = currentTime();     //Hold the end time of play
    if (attackPosition == "S") {                // If save the game
        gSummary->winner = 's';                 // No one win the game, and is
saved game
        int rVal = writeSummary(gSummary, gInfo);   // Write a summary of game
        saveGame(battleShipInfo[0], battleShipInfo[1], playerMatrix, pcMatrix); //
save the game
    } //end if
    else {
        int rVal = writeSummary(gSummary, gInfo);   //Play is ended, write a
summary
    }
    delete gSummary;
} //End startPlay function

//*************************************************************************
//Definition of function checkSuccessful.Use double pointer, number of
//row,col and structure pointer.This function check the attack is successful
//or not, and return Boolean status to update the ship position structure.
//*************************************************************************
bool checkSuccessful(char** pMatrix, BattleShip* sPosition, int row, int col) {
    //Declare variables
    const int AIRCRAFT_LENGTH = 5; // Unit length of the aircraft
    const int BATTLESHIP_LENGTH = 4; // Unit length of the battleship
    const int DESTROYER_LENGTH = 3;  // Unit length of the destroyer
    const int CORVETTE_LENGTH = 2;  // Unit length of the corvette
    bool isFound = false;
    // Check the air craft position
    for (int i = 0; i < AIRCRAFT_LENGTH; i++) {
        if (sPosition->aircraft[i][0] >= 0) {  // If the position not hit yet
            if (sPosition->aircraft[i][0] == row && sPosition->aircraft[i][1] ==
col) //hit successful
            {
```

```
                if (sPosition->aircraft[i][0] == 0) { // If the row index zero;
cannot negative it just multiply -1
                    sPosition->aircraft[i][0] = -10; // Track this position
destroy by setting -10
                } //End if
                else {
                    sPosition->aircraft[i][0] *= -1; // Track this position
destroy by multiplying -1
                }//End else
                pMatrix[row][col] = '@'; // Set the matrix position '@' if hit
successfully
                isFound = true; // The value is found already
                return isFound;
            } //End if
        } //End if
    }//End for-loop

    for (int i = 0; i < BATTLESHIP_LENGTH; i++) { // Check the battle ship
position
        if (sPosition->battleship[i][0] >= 0) { // If the position not hit yet
            if(sPosition->battleship[i][0]==row && sPosition-
>battleship[i][1]==col) //hit successful
            {
                if (sPosition->battleship[i][0] == 0) {
                    sPosition->battleship[i][0] = -10; // Track this position
destroy
                } //End if
                else {
                    sPosition->battleship[i][0] *= -1;// Track this position
destroy
                }
                pMatrix[row][col] = '@';// Set the matrix position '@' if hit
successfully
                isFound = true;// The value is found already
                return isFound;
            }//End if
        }//End if
    } //End for-loop

    for (int i = 0; i < DESTROYER_LENGTH; i++) { // Check the Destroyer position
        if (sPosition->destroyer[i][0] >= 0) {  // If position not hit yet
            if (sPosition->destroyer[i][0] == row &&
                sPosition->destroyer[i][1] == col) { //hit successful

                if (sPosition->destroyer[i][0] == 0) {
                    sPosition->destroyer[i][0] = -10; // Track this position
destroy
                } //End if
                else {
                    sPosition->destroyer[i][0] *= -1; // Track this position
destroy
                } //End else
                pMatrix[row][col] = '@'; // Set the matrix position '@' if hit
successfully
                isFound = true;          // The value is found already
                return isFound;
            } //End if
        } //End if
```

```
    } //End for-loop

    for (int i = 0; i < CORVETTE_LENGTH; i++) { // Check the corvette position
        if (sPosition->corvette[i][0] >= 0) {   // If position not hit yet
            if (sPosition->corvette[i][0] == row && sPosition->corvette[i][1] ==
col) //hit successful
            {
                if (sPosition->corvette[i][0] == 0) {
                    sPosition->corvette[i][0] = -10; // Track this position
destroy
                }//End if
                else {
                    sPosition->corvette[i][0] *= -1; // Track this position
destroy
                }
                pMatrix[row][col] = '@'; // Set the matrix position '@' if hit
successfully
                isFound = true;         // The value is found already
                return isFound;
            }//End if
        }//End if
    } //End for-loop

    pMatrix[row][col] = 'o';
    return isFound;
} // End checkSuccessful function

//*************************************************************************
//Definition of function currentDate
//Input->: None, This function convert the current date as MM-DD-YYYY format
//Output->: Return string sTime
//*************************************************************************
string currentDate() {
    string sTime;
    time_t curr_time;
    curr_time = time(NULL);
    tm* tm_local = localtime(&curr_time);
    sTime = to_string(tm_local->tm_mon + 1) + "/" + to_string(tm_local->tm_mday)
        + "/" + to_string(tm_local->tm_year % 100);
    return sTime;
} // End currentDate function

//*************************************************************
//Definition of function currentTime
//Input: None, This function convert the current time as HH:mm:ss format
//Output: Return string sTime
//*************************************************************
string currentTime() {
    string sTime;
    time_t curr_time;
    curr_time = time(NULL);
    tm* tm_local = localtime(&curr_time);
    sTime = to_string(tm_local->tm_hour) + ":" + to_string(tm_local->tm_min) +
        ":" + to_string(tm_local->tm_sec);
    return sTime;

} //End currentTime function
```

```
//*****************************************************************
//Definition of function writeSummary. Use structure pointer, fstream
//object as @param. This function write game summary in a file
//Return integer value
//*****************************************************************
int writeSummary(GameSummary* gSummary, fstream& gmInfo) {
    gmInfo.open("gameSummary.txt", ios::out);
    gmInfo << gSummary->playerName << ";"
           << setw((gSummary->dt.dateOfPlay).size() + 1);
    gmInfo << gSummary->dt.dateOfPlay << ";";
    gmInfo << setw(gSummary->dt.startTimeOfPlay.size() + 1);
    gmInfo << gSummary->dt.startTimeOfPlay << ";";
    gmInfo << setw(gSummary->dt.endTimeOfPlay.size() + 1);
    gmInfo << gSummary->dt.endTimeOfPlay << ";";
    gmInfo << setw(to_string(gSummary->totalNumberOfAttack).size() + 1);
    gmInfo << gSummary->totalNumberOfAttack << ";";
    gmInfo << setw(2);
    gmInfo << gSummary->winner;
    gmInfo.close();
    return 0;
} // End writeSummary function

//*****************************************************************
//Definition of function saveGame. Use structure variables, double
//pointer as parameters, and write the game in binary file.
//Return none
//*****************************************************************
void saveGame(BattleShip player, BattleShip computer, char** playerMatrix, char**
pcMatrix)
{
    char pMat[ROWS][COLS];
    char cMat[ROWS][COLS];
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            pMat[i][j] = playerMatrix[i][j];
            cMat[i][j] = pcMatrix[i][j];
        }//end for-loop
    } //end for-loop
    fstream pShip, pMatrix, cShip, cMatrix; //Declare fstream objects to write out
    //open multiple binary files for output
    pShip.open("playerShip.dat", ios::out | ios::binary);
    cShip.open("computerShip.dat", ios::out | ios::binary);
    pMatrix.open("playerMatrix.dat", ios::out | ios::binary);
    cMatrix.open("pcMatrix.dat", ios::out | ios::binary);
    //write structure data to the binary file
    pShip.write(reinterpret_cast<char*>(&player), sizeof(player));
    cShip.write(reinterpret_cast<char*>(&computer), sizeof(computer));
    //write matrix data to the binary file
    pMatrix.write(reinterpret_cast<char*>(&pMat), (ROWS * COLS) * sizeof(char));
    cMatrix.write(reinterpret_cast<char*>(&cMat), (ROWS * COLS) * sizeof(char));
    //close the file
    pShip.close();
    cShip.close();
    pMatrix.close();
    cMatrix.close();
} //End saveGame function

//*****************************************************************
```

```
//Definition of function openGame.
//Input->:None, and read the game from binary file.
//Output->:Return Boolean status
//*****************************************************************
bool openGame() {
    char plMatrix[ROWS][COLS]; //To read in from binary file to 2d array
    char coMatrix[ROWS][COLS]; //To read in from binary file to 2d array
    BattleShip bsPlayer[2];   //To read in from binary file to structure variable
    fstream pMatrix, cMatrix, pShip, cShip; //Declare fstream object to read in
    string playerName; //Name of player

    //open multiple binary files to read in
    pShip.open("playerShip.dat", ios::in | ios::binary);
    cShip.open("computerShip.dat", ios::in | ios::binary);
    pMatrix.open("playerMatrix.dat", ios::in | ios::binary);
    cMatrix.open("pcMatrix.dat", ios::in | ios::binary);
    if (pMatrix.fail() || cMatrix.fail() || pShip.fail() || cShip.fail())
        return false;
    //Read in from binary files
    pShip.read(reinterpret_cast<char*>(&bsPlayer[0]), sizeof(bsPlayer[0]));
    cShip.read(reinterpret_cast<char*>(&bsPlayer[1]), sizeof(bsPlayer[1]));
    pMatrix.read(reinterpret_cast<char*>(&plMatrix), (ROWS * COLS) *
sizeof(char));
    cMatrix.read(reinterpret_cast<char*>(&coMatrix), (ROWS * COLS) *
sizeof(char));
    //close the files
    pMatrix.close();
    cShip.close();
    pMatrix.close();
    cMatrix.close();
    // Send file fstream object as a function reference parameter
    bool fOpen = openSummaryFile(pMatrix, "gameSummary.txt");
    if (fOpen) {
        string playerName = readContent(pMatrix);
        pMatrix.close();

        //Display the ship status
        int retInit = displayShipStatus(bsPlayer[0], bsPlayer[1]);
        //Display the player information and game head
        gameHead(playerName);
        char** playerMatrix;
        char** pcMatrix;
        playerMatrix = new char* [ROWS]; //Allocating the row space in heap
dynamically
        pcMatrix = new char* [ROWS]; //Allocating the row space in heap
dynamically
        for (int i = 0; i < ROWS; i++) {
            playerMatrix[i] = new char[COLS];  //Allocating the column space in
heap dynamically
            pcMatrix[i] = new char[COLS];  //Allocating the column space in heap
dynamically
        }
        for (int i = 0; i < ROWS; i++) {
            for (int j = 0; j < COLS; j++) {
                playerMatrix[i][j] = plMatrix[i][j];
                pcMatrix[i][j] = coMatrix[i][j];
            }
        }
```

```
        cin.ignore();
        //Display the player and computer matrix
        showPlayZone(playerMatrix, pcMatrix);
        // Start the play allow player and computer to play
        startPlay(playerName, playerMatrix, pcMatrix, bsPlayer);
        destroy(playerMatrix, ROWS, COLS); //Clear the memory
        destroy(pcMatrix, ROWS, COLS); //Clear the memory
    } //end if
    else {
        cout << "Save game is not available now!";
    } //end else
    return true;
} //End openGame function

//*****************************************************************
//Definition of function openSummaryFile. Use fstream object
//and string parameters.This function check file open in or not.
//Return Boolean type status
//*****************************************************************
bool openSummaryFile(fstream& infile, string fileName) {
    infile.open(fileName, ios::in);
    if (infile.fail()) return false;
    else              return true;

} //End openSummaryFile function

//*****************************************************************
//Definition of function lastWinner.
//Input->:None, and read text file to get the name of winner
//Output->:Return string winner
//*****************************************************************
string lastWinner() {
    char ch; //to get a char from text file
    string winner = ""; //to store name of winner

    fstream file("gameSummary.txt", ios::in); //open file to read in
    if (file.fail()) {//Check file open in or not
        cout << "No information available now" << endl;
        return "";
    }
    file.seekg(-1L, ios::end);
    file.get(ch);//get a char from file
    if (ch == 'p') {
        winner = "Human";
    }
    else if (ch == 'c') {
        winner = "Computer";
    }
    else {
        winner = "None-saved game";
    }
    file.close();
    return winner;
} //End lastWinner function

//*****************************************************************************
//Definition of function lastGameSummary.
//Input->:None, this function write and read text file for last game summary
```

```
//Output->:Return none
//*************************************************************************
void lastGameSummary() {
    //Declare variable
    string delimiter = ";"; // For split the line
    string name = "";       // To hold name of the player
    char yn;                // ask to get input for updating or not
    char ch;                // To get a char from file
    string sInfo[5];        // To hold info from file
    string line;            // To hold line from file
    int i = 0;              // To track the sInfo array index
    size_t pos = 0;         // To hold the size of sub-string

    //Open text file to write and read in
    fstream file("gameSummary.txt", ios::in | ios::out);
    if (file.fail()) {
        cout << "No summary available now" << endl;
        return;
    }//end if
    //Read from file to string
    getline(file, line);
    // Split the line
    while ((pos = line.find(delimiter)) != std::string::npos) {
        sInfo[i] = line.substr(0, pos);
        line.erase(0, pos + delimiter.length());
        i++;
    }//end while-loop
    //Display the output to the screen
    cout << "\t\t Player name: " << *sInfo << endl; // Used pointer notation
instead of subscripts
    cout << "\t\t Date: " << *(sInfo + 1) << endl;
    cout << "\t\t Start time: " << *(sInfo + 2) << endl;
    cout << "\t\t End time: " << *(sInfo + 3) << endl;
    cout << "\t\t Total attack: " << *(sInfo + 4) << endl;
    cout << "\t\t Do you want update the player name: (y/n) ";
    cin.ignore();
    cin >> yn;
    //update to the output file
    if (yn == 'y') {
        cin.ignore();
        cout << "Please input the name for updating: " << endl;
        getline(cin, name);
        file.seekg(-1L, ios::end);
        file.get(ch);
        file.seekp(0L, ios::beg);
        file << name << ";" << setw(name.size() + 1);
        file << sInfo[1] << ";" << setw(sInfo[1].size() + 1);
        file << sInfo[2] << ";" << setw(sInfo[2].size() + 1);
        file << sInfo[3] << ";" << setw(sInfo[4].size() + 1);
        file << sInfo[4] << ";" << setw(2);
        file << ch;

    }//end if
    file.close();//close the file
} //End lastGameSummary function

//*************************************************************************
//Definition of function readContent.
```

```
        //Input->:fstream object as @param, function find the last player name from file
        //Output->:Return token as player name
        //*****************************************************************************
        string readContent(fstream& infile) {
            string line;            //To read line from file
            string delimiter = ";"; //To read line until this delimiter
            string token;           //To hold player name
            infile >> line;         //Read line from file
            //Find name and store to the token
            token = line.substr(0, line.find(delimiter));
            return token;   //Return the player name
            infile.close(); //Close the file
        } // End readContent function

        //*****************************************************************
        //Definition of function destroy                                  *
        //De-allocate memory.                                             *
        //*****************************************************************
        void destroy(char** array, int rows, int cols) {
            //Free the memory after the use of array
            for (int i = 0; i < rows; i++) {
                delete[] array[i];
            }
}
```