Project-2

Title

Battleship Game

Course

CSC 17A

Section

48290

Due Date

12/04/2024

Author

Khadiza Akter

Contents

1. Introduction:

Battleship is a two-player strategy game, also known as Battleships or Sea Battle. Each player positions a fleet of warships on designated grids, either on paper or a board, without the opponent seeing their locations. The objective is to destroy the opposing player's fleet by alternating turns, where each player attempts to hit their opponent's ships by calling out specific coordinates.

The game is played on two sets of grids, one for each player. These grids are usually square, most commonly 10×10 in size, with each square identified by a combination of a letter and a number. When a player hits an opponent's ship, the hit is marked with a different symbol or letter to indicate success.

Before the game begins, each player secretly places their ships on their own grid. Each ship is arranged in a series of adjacent spaces, either horizontally or vertically, with the number of spaces occupied depending on the type of ship. Ships cannot overlap, meaning no two ships can occupy the same grid position. Both players have the same types and numbers of ships. The ships are hidden from the opponent's view, and players are not allowed to see each other's placements. The objective is to discover and destroy the opponent's fleet first, with the player who eliminates all enemy ships emerging as the winner [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 five different versions.

GameOfBattleShip_V1:

In version 1, the project was started and filled-up the 2-D list for player and computer, and developed the function to display the game rules.

GameOfBattleShip _V2:

In version 2, the player was able to setup the ships in his/her matrix position.

GameOfBattleShip_V3:

In version 3, the computer was able to setup the ships randomly.

GameOfBattleShip_V4 (Final):

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.

Also this version, 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 for computer and player and displayed status information at the top of the screen. Finally, this version determined the winner of the game.

Lines of code: 1593 (Including Spaces and Comments).

I worked on the project for around four weeks and spent around 180 hours. I applied several concepts from recurssion, graph and tree to use these concepts in a software project. Apart from these chapters, I had to add concepts such as random variables, operator overloading.

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.

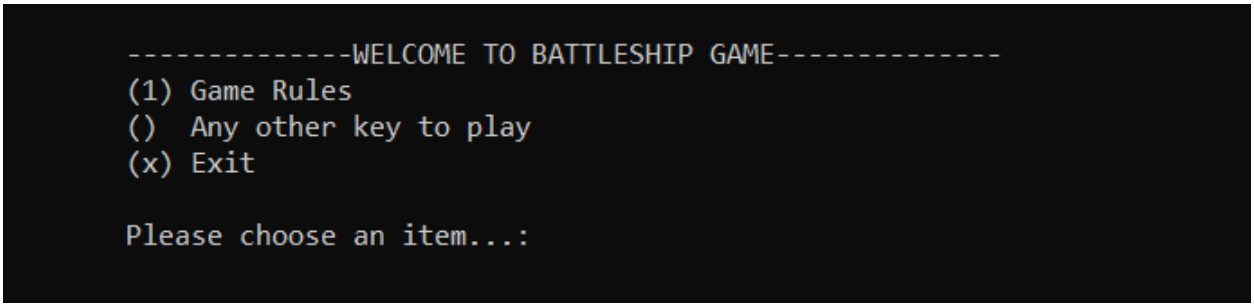


Figure 1. Game Menu

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

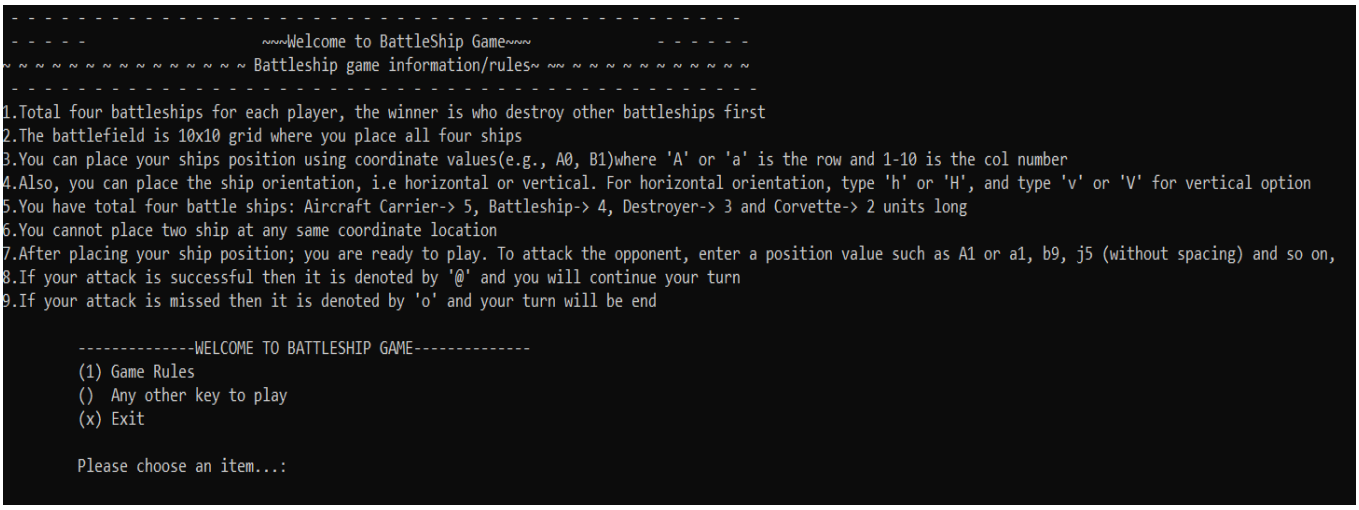


Figure 2. Game information/rules

Press ‘x’ for exit the game now.

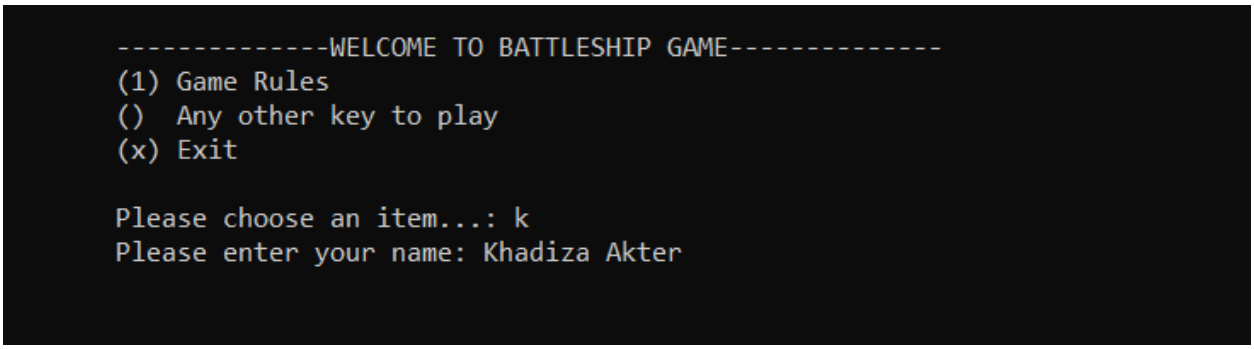


Figure 3. A new game start

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

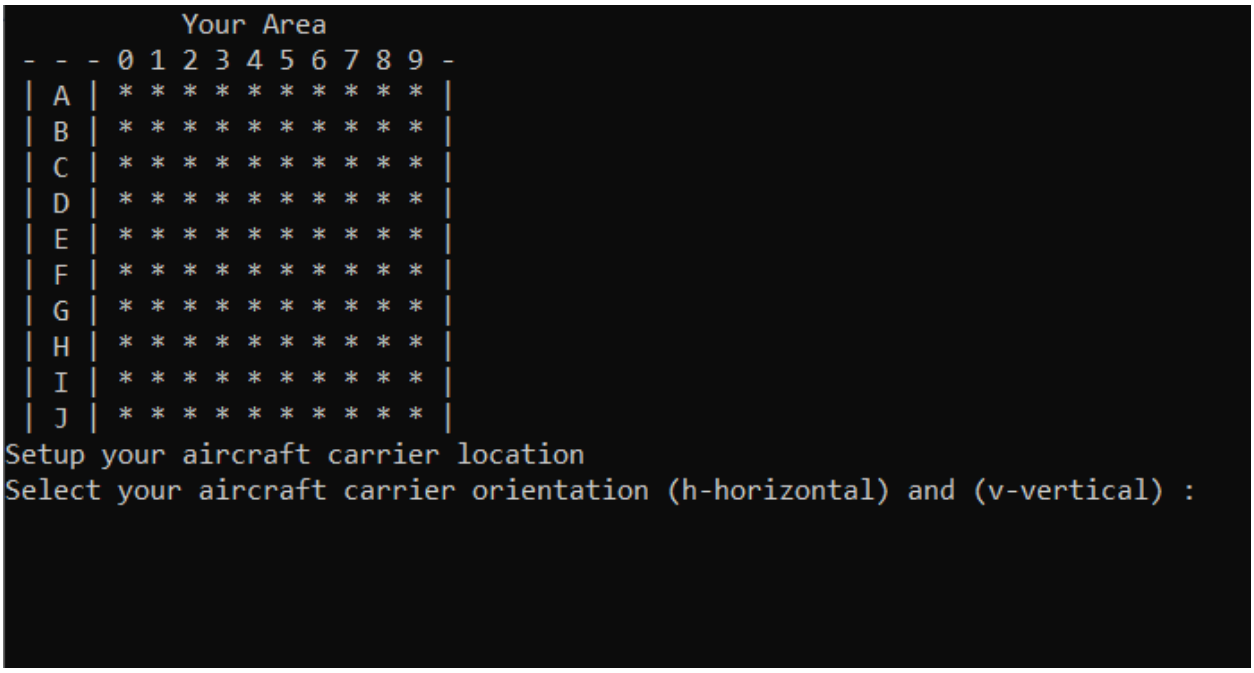


Figure 4. Setup player battle ships

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.

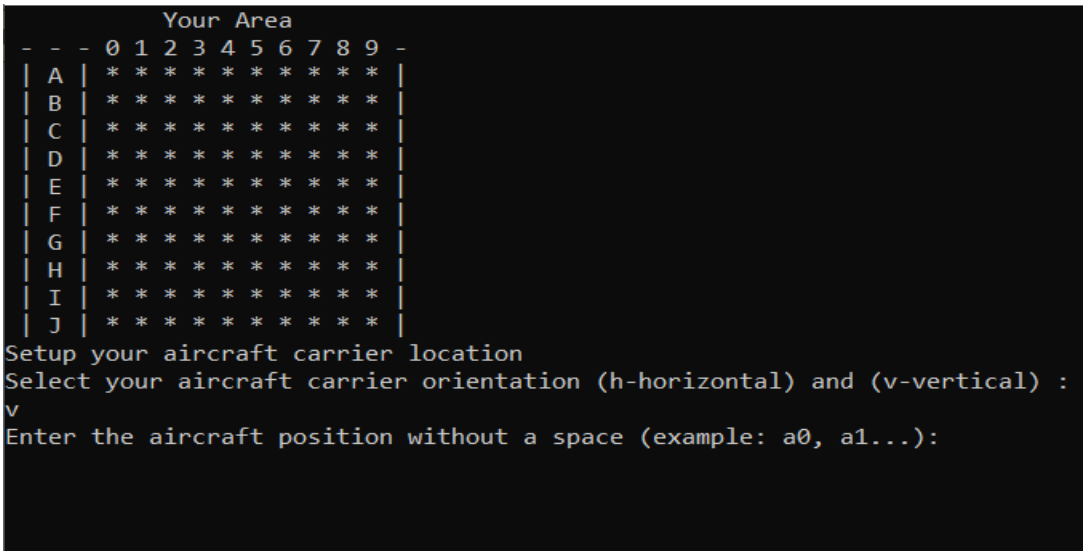


Figure 5. Aircraft setup

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.

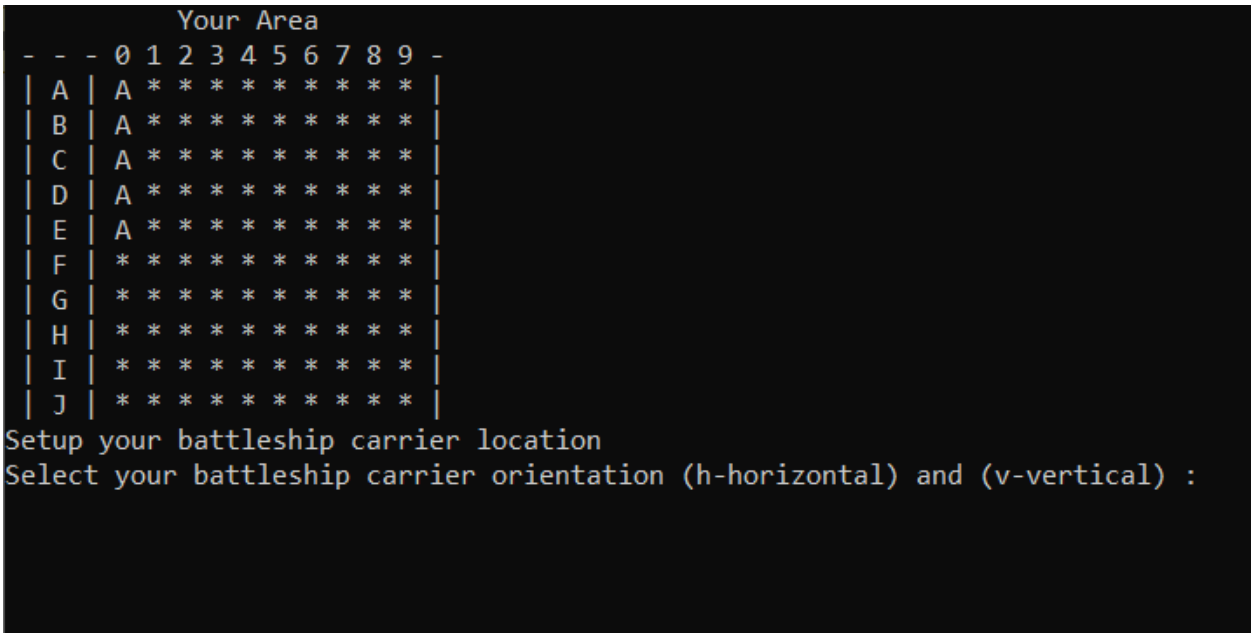


Figure 6. Setup aircraft as ‘h’ and position ‘a0’

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.

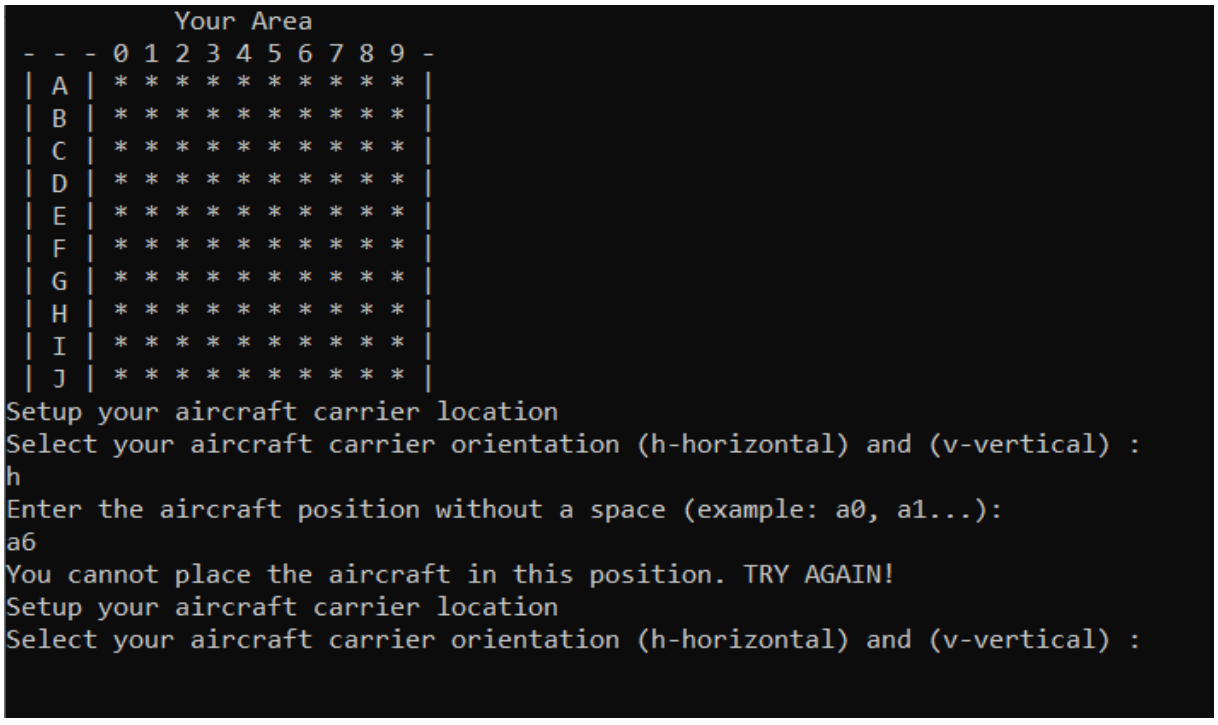


Figure 7. Setup aircraft as ‘h’ and position ‘a6’

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.

```
.....................................................................
 -  Computer Ships Status                  Your Ship Status-
.....................................................................
----Aircraft: 5 units                  | ----Aircraft: 5 units
----Battleship: 4 units                | ----Battleship: 4 units
----Destroyer: 3 units                 | ----Destroyer: 3 units
----Corvette: 2 units                  | ----Corvette: 2 units
.....................................................................
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -                    ~~~Welcome to BattleShip~~~                   -
 -                    ~~~Player Name: Khadiza Akter
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -          Computer Zone                              Your Zone      -
 - - - 0 1 2 3 4 5 6 7 8 9 -           - - - 0 1 2 3 4 5 6 7 8 9 -
 | A | * * * * * * * * * * |           | A | A * * * * * D * * * |
 | B | * * * * * * * * * * |           | B | A * * * * * D * * * |
 | C | * * * * * * * * * * |           | C | A * * * * * D * * * |
 | D | * * * * * * * * * * |           | D | A * * B B B B * * * |
 | E | * * * * * * * * * * |           | E | A * * * * * * * * * |
 | F | * * * * * * * * * * |           | F | * * * * * * * * * * |
 | G | * * * * * * * * * * |           | G | * * * C * * * * * * |
 | H | * * * * * * * * * * |           | H | * * * C * * * * * * |
 | I | * * * * * * * * * * |           | I | * * * * * * * * * * |
 | J | * * * * * * * * * * |           | J | * * * * * * * * * * |
~~ Now your turn to attack the computer ship position ~~
Choose a position for attacking the computer ships (example: a0, a1...):
```

Figure 8. Game playing screen

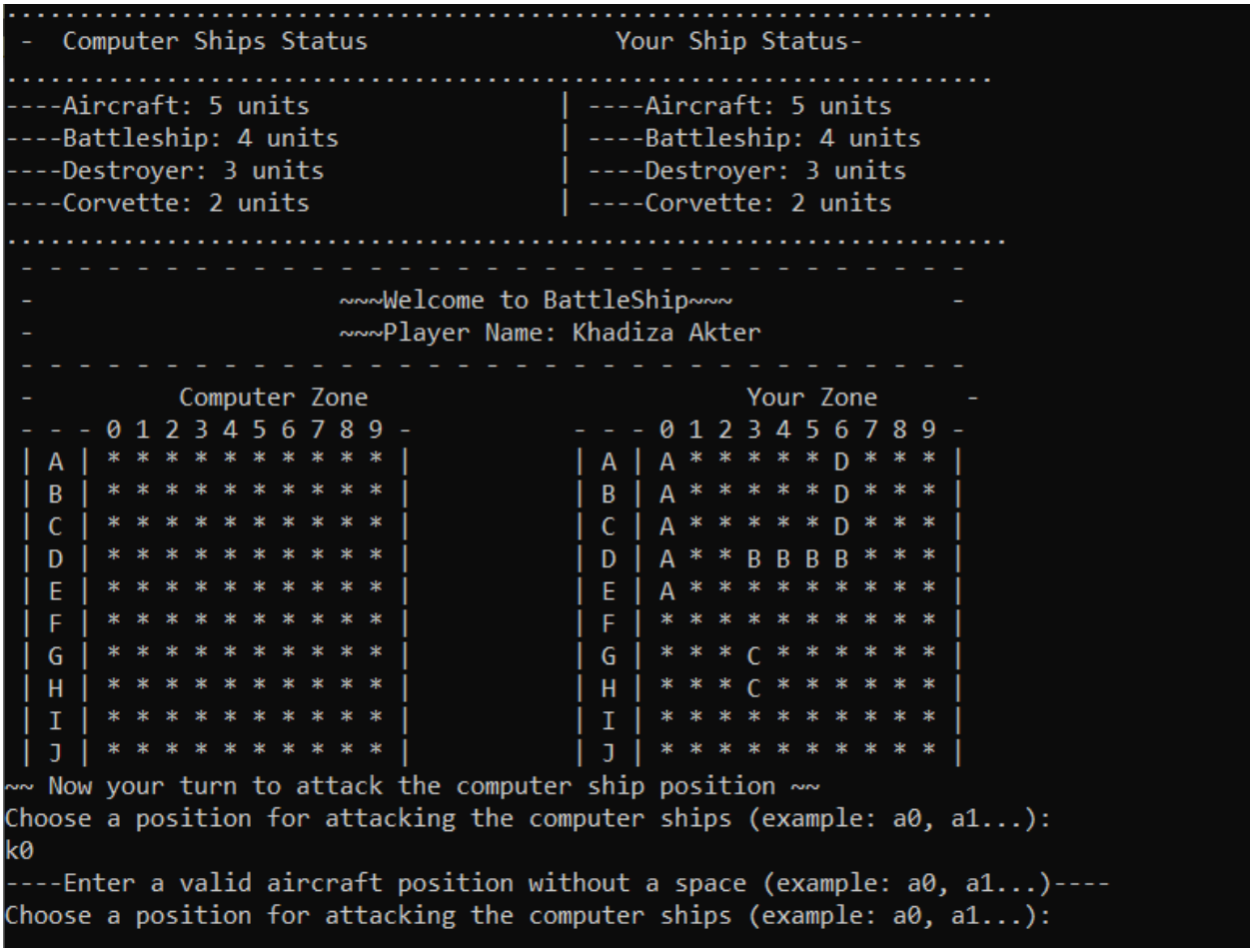


Figure 9. Error input and asking for input position again

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.

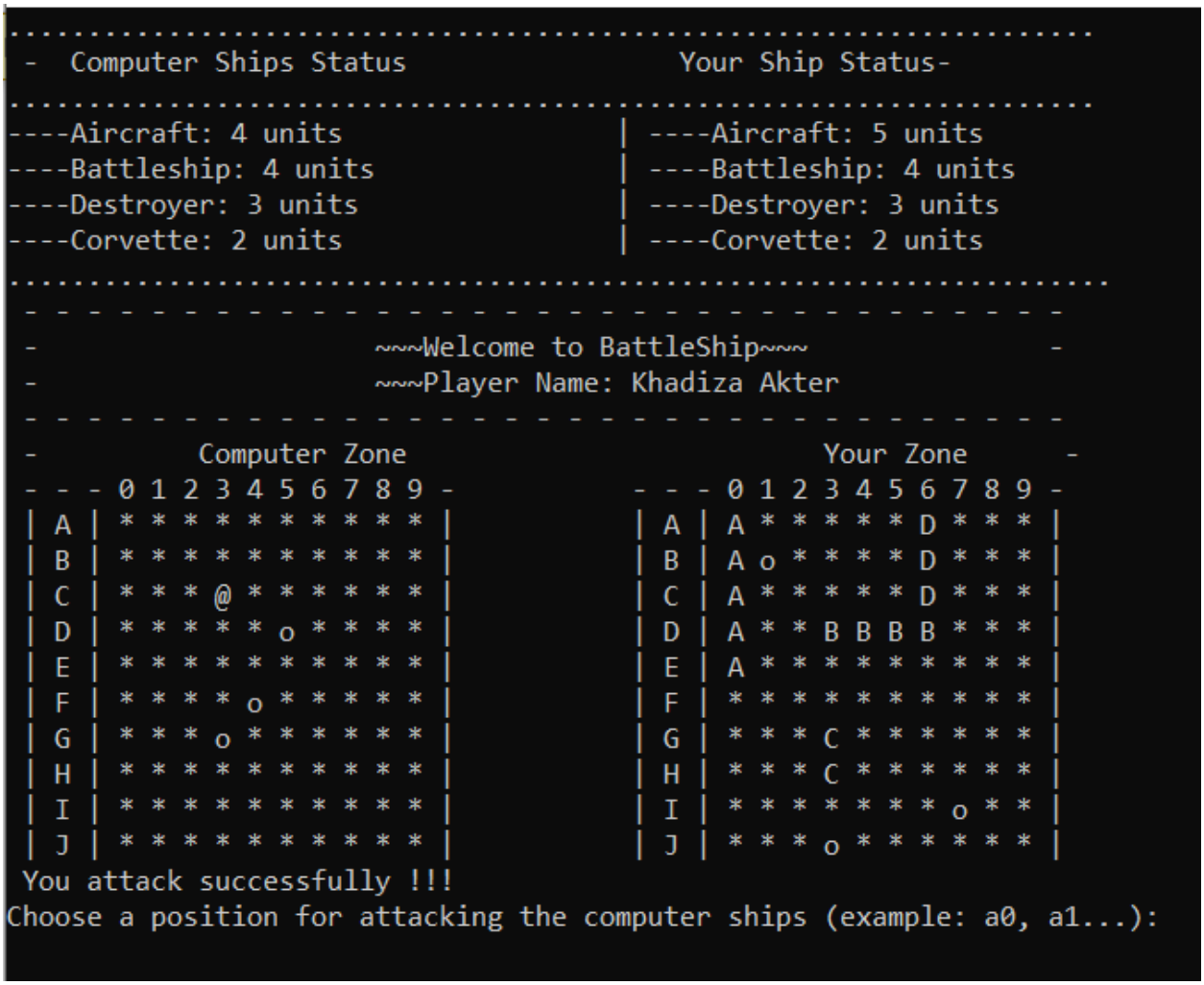


Figure 10. Player attack successful and ask for position again to hit

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

Figure 13. Player won the game and exit

3.3 Specifications of the game

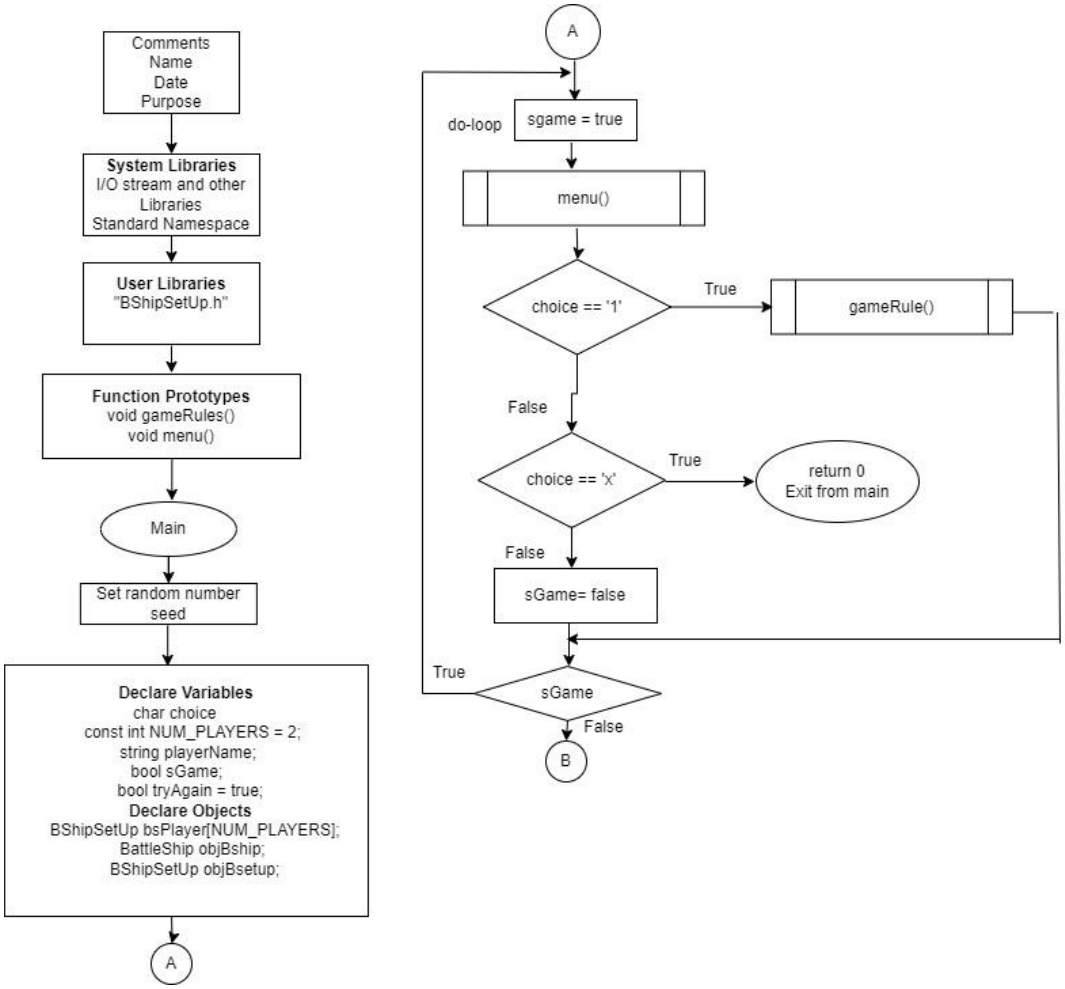
3.3.1 UML Diagram

The overall UML diagram of the project is given as follow:

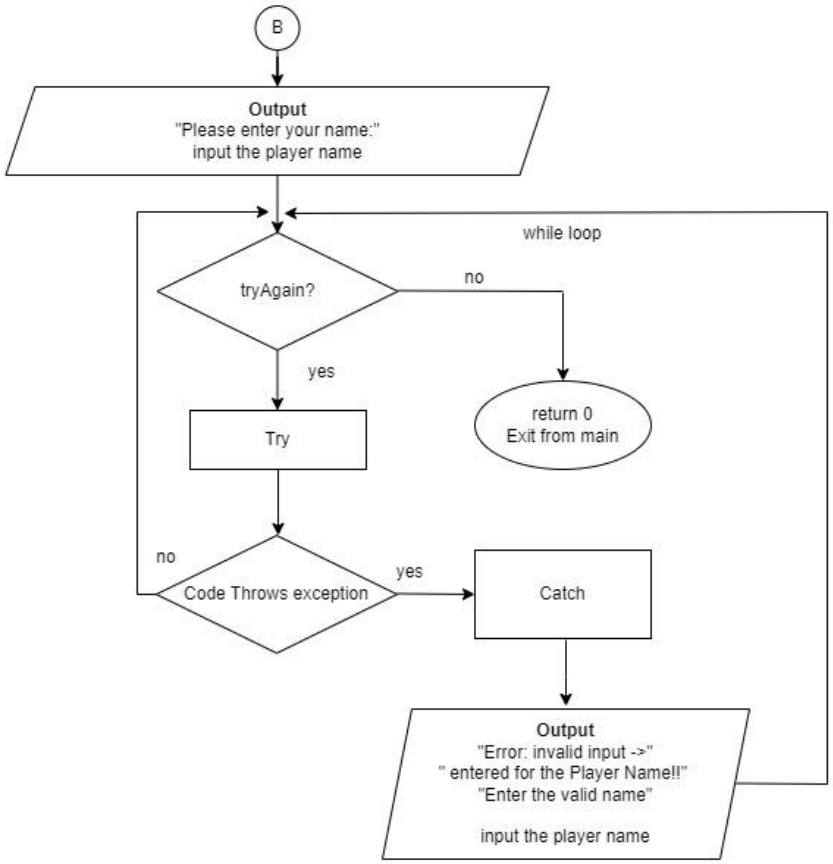
Figure 17. UML Diagram of the game

3.3.2 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:

Create the object for BattleShip, BShipSetUp

repeat

set sGame = true

draw the menu

choice a menu item

if choice is 1 then

Show the game rules

else if choice is 'x' then

Exit the game

else

Set sGame = false

end if

Until sGame is false

Input the player name

While tryAgain do

Try

Setup player name using BShipSetUp constructor

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 both player and computer zones

Start the play, win, or loss the game

Set tryAgain = false

Catch Exception

Exit the game

3.4 Checkoff Sheet

Recursions:

File: BShipSetup.cpp
Function Name: void BShipSetUp::drawRow(list<list<char>>::iterator it, list<list<char>>::iterator end, char yCoord[], int index)
Location: line # 74
This function to draw rows of the grid recursively.

Function Name: void BShipSetUp::drawElements(list<char>::iterator it, list<char>::iterator end)
Location: line # 96
This function to draw elements of a row recursively

Recursive Sort:

File: BShipSetup.cpp
Function Name: void BShipSetUp::quickSort(int arr[], int low, int high)

Location: Line # 177

This function used to sort the sizes of battle ship.

Hashing:

Function Name: int letterToRowNumber(char letter)
Location: Line # 112-115
The hashing used to convert the letter to row number

Trees:

Function Name: void BShipSetUp::showPlayZone()
Line # 838
Tree Related functions:

BShipSetUp::TreeNode* BShipSetUp::insert(TreeNode* root, char value), Line # 813;
void BShipSetUp::inorderTraversal(TreeNode* root, list<char>& nodes),Line # 796

The tree was used to draw both computer and player y-coordinate values

Graph:

void BShipSetUp::startPlay(BShipSetUp battleShipPlayer, BShipSetUp battleShipComputer), line # 880;

Related files: Graph.h and Graph.cpp
The graph was used to find the min and max value whether a player provides a valid move.

REFERENCES:

Learn the battleship game:

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

[2] https://stackoverflow.com/

Draw the diagram

[3] https://doxygen.nl/

Program Listing:

```
/*
 * File:   main.cpp
 * Author: Khadiza Akter
 * Created on October 07, 2024, 10:24 PM
 * Purpose: Game of BattleShip
 *          Fill 2-D list with '*'
 *          Apply the Rules of Battleship
 *          Set up player battleship
 *          Set up computer battleship
 *          Draw the both matrix and playing zone
 *          Start the play, and allow the user input and computer
            randomly selection input
 *          Specification and implementation of Battleship(base) and
            BShipSetUp(derived)classes
            Winner of the Game

 */
 //System Level Libraries
#include <iostream> //Input-output library
#include <string>   //Strings
#include <cstdlib>  //Srand to set the seed or system()
#include <ctime>    //Set for time()
using namespace std;//Standard Name-space under which System Libraries reside

//User defined libraries
#include "BShipSetUp.h" //needed for BShipSetUp class

//Function Prototypes
void gameRules();   //display the game rules
//Execution begins here!
int main(int argc, char** argv) {
    //Random seed here
    srand(static_cast<unsigned int>(time(0)));
    //Declare variables
    char choice;       // take input for checking the start play or game rules
    const int NUM_PLAYERS = 2; // Number of player
    BShipSetUp bsPlayer[NUM_PLAYERS];// Array of class BShipSetUp
    string playerName;   //To hold the player name
    bool sGame;         // To track do while-loop for menu
    bool tryAgain = true; //Flag to reread the input name


    //Output the game statistics or menu to the screen
    do {
        sGame = true;
        cout << "\n\t --------------WELCOME TO BATTLESHIP GAME--------------\n";
        cout << "\t (1) Game Rules " << endl;
        cout << "\t ()  Any other key to play " << endl;
        cout << "\t (x) Exit " << endl;
        cout << "\n\t Please choose an item...: ";

        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 == '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 << "\t Please enter your name: "; //Ask user to enter name
    getline(cin, playerName); // Take the player name

    while (tryAgain) {
        try {
            //create an object of BShipSetUp class and using constructor
            //to initialize member
            BShipSetUp bSetUp(playerName);
            //Display player information
            bSetUp.gameHead();
            //Display player matrix
            bSetUp.drawPlayerArea(bSetUp.getMatrixData());
            //Set player ship position and return structure
            bsPlayer[0] = bSetUp.setYourBattleShip(bSetUp);
            //Set computer ship position and return structure
            bsPlayer[1] = bSetUp.setComputerBattleShip(bSetUp);
            int retInit = bSetUp.displayShipStatus(bsPlayer[0], bsPlayer[1]);
            bSetUp.gameHead();
            bSetUp.showPlayZone();
            bSetUp.startPlay(bsPlayer[0], bsPlayer[1]);
            tryAgain = false;
        }  //End try
        catch (BShipSetUp::InvalidName) {
            cout << "\t Error: invalid input -> [" << playerName
                << "] entered for the Player Name!!" << endl;
            cout << "\t Please enter a valid name: ";
            getline(cin, playerName); // Take the player name

        } //End catch
    } //End while-loop

    //Exit the program
    return 0;
} //end of main function

//**************************************************************
//Definition of function gameRules
//Input->: None, data on game rules
//Output->:No return, Display the game rules
//**************************************************************
void gameRules() {

    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



    /*
     * File: BShipSetup.cpp
     * Author: Khadiza Akter
     * Created on October 10, 2024, 10:02 PM
     * Purpose: Implementation file for BShipSetUp class
     */

    #include <iostream> //Input-output library
    #include <cstdlib>  //Srand to set the seed or system()
    #include <ctime> //set for time
    #include <string>
    #include <cstring>
    #include <list>
    #include <map>
    #include <array>
    #include <algorithm>
    #include <unordered_map>

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

    #include "BShipSetUp.h"
    #include "Graph.h"

    //*****************************************************************
    //Definition of function gameHead. This is a member function of   *
    //BShipSetUp class and display the player name
    //*****************************************************************
    void BShipSetUp::gameHead() {
        //system("cls");
        cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n";
        cout << " -                     ~~~Welcome to BattleShip~~~              -\n";
        cout << " -                     ~~~Player Name: " << playerName << endl;
        cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n";
    }
    //*********************************************************************
    //Definition of operator= . This function overloaded = operator
    //when assign one object to another object.
    //Return constant BShipSetUp object.
    //*********************************************************************
```

```
const BShipSetUp BShipSetUp::operator=(const BShipSetUp& right) {
    if (this != &right) {
        memcpy(aircraft, right.aircraft, 5 * 2 * sizeof(int));
        memcpy(battleship, right.battleship, 4 * 2 * sizeof(int));
        memcpy(destroyer, right.destroyer, 3 * 2 * sizeof(int));
        memcpy(corvette, right.corvette, 2 * 2 * sizeof(int));
    }
    return *this;
}


//***************************************************************
//Definition of function drawPlayerArea                         *
//Display player matrix                                         *
//***************************************************************
// Main function to draw the player's area
void BShipSetUp::drawPlayerArea(list<list<char>> matrixData) {
    // Clear the screen
    system("cls");

    // Display the title and x-coordinates
    cout << "           Your Area                          " << endl;
    cout << " - - - 0 1 2 3 4 5 6 7 8 9 -            " << endl;

    // Define the y-coordinates (row labels) for the grid
    char yCoord[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', '\0' };

    // Call the recursive function to draw each row
    drawRow(matrixData.begin(), matrixData.end(), yCoord, 0);
}
//***************************************************************
//Definition of function drawRow.
// Function to draw rows of the grid recursively
//***************************************************************
void BShipSetUp::drawRow(list<list<char>>::iterator it, list<list<char>>::iterator end,
char yCoord[], int index) {
    // Base case: Stop if there are no more rows
    if (it == end) {
        return;
    }

    // Print the y-coordinate for the current row (like "A", "B", etc.)
    cout << " | " << yCoord[index] << " | ";

    // Call a recursive function to draw the elements of the current row
    drawElements((*it).begin(), (*it).end());

    // Finish the row with a closing border
    cout << "|" << endl;

    // Move to the next row (increment index and iterator) and call recursively
    drawRow(next(it), end, yCoord, index + 1);
}
//***************************************************************
//Definition of function drawElements.
// Function to draw elements of a row recursively
//***************************************************************
void BShipSetUp::drawElements(list<char>::iterator it, list<char>::iterator end) {
    // Base case: Stop if there are no more elements in the row
    if (it == end) {
        return;
```

```
    }
    // Print the current element
    cout << *it << " ";

    // Move to the next element and call recursively
    drawElements(next(it), end);
}
//*************************************************************************
//Definition of function letterToRowNumber.This function is declared as a
*
//friend by Battleship class and it will determine the letter (A, B, C...J)
// value to integer y-axis value (0,1,2...9) using hashTable
//*************************************************************************
int letterToRowNumber(char letter) {
    // Define a hash table as an array of integers
    const int hashTableSize = 26; // 26 letters in the English alphabet
    int hashTable[hashTableSize];

    // Initialize all positions to -1 (invalid)
    for (int i = 0; i < hashTableSize; ++i) {
        hashTable[i] = -1;
    }

    // Map 'A' to 0, 'B' to 1, ..., 'J' to 9
    for (int i = 0; i <= 9; ++i) {
        hashTable['A' - 'A' + i] = i; // Offset based on 'A'
    }

    // Compute hash key
    int index = letter - 'A';
    if (index >= 0 && index < hashTableSize) {
        return hashTable[index];
    }
    else {
        return -1; // Invalid selection
    }
}

//*************************************************************
//Definition of function conflictWithOtherShip                    *
//This check the ship position conflict with other ship or not    *
//*************************************************************
bool BShipSetUp::conflictWithOtherShip(list<std::list<char>> playerMatrix, int row, int
col,
        int shipLength, char shipOrientation)
{
    if (shipOrientation == 'h')                     // check the ship orientation
    {
        auto rowIt = std::next(playerMatrix.begin(), row);
        for (int i = col; i < col + shipLength; i++) // for horizontal orientation
check the column till ship length
        {
            auto k = std::next(rowIt->begin(), i);
            if (*k != '*') {       // check the character for position of the matrix,
if it is not '*' that means it is conflict with other ship position
                return true;                        // and return true
            }
        }
    }
    else
    {
```

```
        //auto colIt = std::next(playerMatrix.begin(), col);
        for (int i = row; i < row + shipLength; i++)  // for horizontal orientation
check the row till ship length
        {
            auto rowIt = std::next(playerMatrix.begin(), i);
            auto colIt = std::next(rowIt->begin(), col); // Accessing the column

            //auto k = std::next(colIt->begin(), row);
            if (*colIt != '*') {       // check the character for position of the
matrix, if it is not '*' that means it is conflict with other ship position
                return true;                          // and return true
            }

        }
    }
    return false;
}

//***************************************************************
//Definition of function QuickSort                    *
//This function sorted the size of battle ships    *
//***************************************************************
void BShipSetUp::quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Choose the last element as the pivot
        int pivot = arr[high];
        int i = low - 1;

        // Partitioning process: elements less than pivot go to the left, greater to
the right
        for (int j = low; j < high; ++j) {
            if (arr[j] > pivot) {
                ++i; // Increment the smaller element index
                swap(arr[i], arr[j]); // Swap current element with the element at i
            }
        }
        // Place the pivot in its correct position
        swap(arr[i + 1], arr[high]);
        int partitionIndex = i + 1;

        // Recursive calls to sort the subarrays
        quickSort(arr, low, partitionIndex - 1);   // Sort elements before the pivot
        quickSort(arr, partitionIndex + 1, high); // Sort elements after the pivot
    }
}
//***************************************************************
//Definition of function setYourBattleShip                      *
//Set player ship position and return structure                  *
//***************************************************************
BShipSetUp BShipSetUp::setYourBattleShip(BShipSetUp b) {
    int size = sizeof(shipSizes) / sizeof(shipSizes[0]); // Calculate the size of the
array
    // Perform QuickSort on the ship sizes array
    quickSort(shipSizes, 0, size - 1);
    const int AIRCRAFT_LENGTH = shipSizes[0];     // Unit length of the aircraft
    const int BATTLESHIP_LENGTH = shipSizes[1];   // Unit length of the battleship
    const int DESTROYER_LENGTH = shipSizes[2];    // Unit length of the destroyer
    const int CORVETTE_LENGTH = shipSizes[3];     // 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)
    BattleShip obj;                    // Declare an object of BattleShip

    //BShipSetUp b;
    playerMatrix = getMatrixData();
    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 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;
            }
            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 aircraft position without a space "
                "(example: a0, a1...): " << endl;
        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
                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
                }
                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;
            }
        }

        if (tolower(shipOrientiation) == 'h') { // check for horizontal setup
```

```
            int startPositionRow = letterToRowNumber(shipPosition[0]);// Get the start
value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';
            if (startPositionCol > AIRCRAFT_LENGTH) { // not able to setup the aircraft
horizontally from this position
                cout << "You cannot place the aircraft in this position. TRY AGAIN!" <<
endl;
                shipPosition.clear();
                continue;
            } //end if
            else {
                int counter = 0;
                auto rowIt = std::next(playerMatrix.begin(), startPositionRow); //
Accessing the row
                for (int i = startPositionCol;
                        i < startPositionCol + AIRCRAFT_LENGTH; i++) {
                    auto colIt = std::next(rowIt->begin(), i);
                    *colIt = 'A';    // set the player matrix with 'A' for indicating
the aircraft location
                    b.aircraft[counter][0] = startPositionRow;  // insert the ship
position values in the structure variable
                    b.aircraft[counter][1] = i;
                    counter++;  // increase the counter one
                }
                break;
            } //end else
        } //end if
        if (tolower(shipOrientiation) == 'v') { // check for the vertical setup
            int startPositionRow = letterToRowNumber(shipPosition[0]);      // Get the
start value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';  // make a character value to
integer

            if (startPositionRow > AIRCRAFT_LENGTH) { // not able to setup the aircraft
vertically from this position
                cout << "You cannot place the aircraft in this position."
                        " TRY AGAIN!" << endl;
                shipPosition.clear();
                continue;
            } //end if
            else {
                int counter = 0;
                for (int i = startPositionRow;
                        i < startPositionRow + AIRCRAFT_LENGTH; i++)
                {
                    auto rowIt = std::next(playerMatrix.begin(), i);
                    auto colIt = std::next(rowIt->begin(), startPositionCol); //
Accessing the column
                    *colIt = 'A';
                    //playerMatrix[i][startPositionCol] = 'A'; // set the player matrix
with 'A' for indicating the aircraft location
                    b.aircraft[counter][0] = i;              // insert the ship
position values in the structure variable
                    b.aircraft[counter][1] = startPositionCol;
                    counter++; // increase the counter one
                } //end for
                break;
            } //end else

        } //end if
    } //end while-loop
```

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

        } //end while-loop

        if (tolower(shipOrientiation) == 'h')   // check for horizontal setup
        {
            int startPositionRow = letterToRowNumber(shipPosition[0]); // Get the start
value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';
```

```
            bool isConflict = conflictWithOtherShip(playerMatrix,
                                startPositionRow, startPositionCol,
                                BATTLESHIP_LENGTH, tolower(shipOrientiation)); // check
the ship is conflict with other ship position
            if ((startPositionCol > 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!" << endl;
                shipPosition.clear();
                continue;
            }
            else
            {
                int counter = 0;
                auto rowIt = std::next(playerMatrix.begin(), startPositionRow); //
Accessing the row
                for (int i = startPositionCol;
                        i < startPositionCol + BATTLESHIP_LENGTH; i++)
                {
                    auto colIt = std::next(rowIt->begin(), i);
                    *colIt = 'B';    // set the player matrix with 'A' for indicating
the aircraft location
                    //playerMatrix[startPositionRow][i] = 'B';    // set the player
matrix with 'B' for indicating the battle location
                    b.battleship[counter][0] = startPositionRow;   // insert the ship
position values in the structure variable
                    b.battleship[counter][1] = i;
                    counter++; // increase the counter one
                } //end for
                break;
            } //end else
        } //end if
        if (tolower(shipOrientiation) == 'v')   // check for the vertical setup
        {
            int startPositionRow = letterToRowNumber(shipPosition[0]); // Get the start
value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';             // make a
character value to integer
            bool isConflict = conflictWithOtherShip(playerMatrix,
                  startPositionRow, startPositionCol, BATTLESHIP_LENGTH,
                  tolower(shipOrientiation)); // check the ship is conflict with other
ship position
            if ((startPositionRow > 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!" << endl;
                shipPosition.clear();
                continue;
            }
            else {
                int counter = 0;
                for (int i = startPositionRow; i < startPositionRow
                        + BATTLESHIP_LENGTH; i++)
                {
                    auto rowIt = std::next(playerMatrix.begin(), i);
                    auto colIt = std::next(rowIt->begin(), startPositionCol); //
Accessing the column
                    *colIt = 'B';
```

```
                    //playerMatrix[i][startPositionCol] = 'B';  // set the player
matrix with 'B' for indicating the battle location
                    b.battleship[counter][0] = i;            // insert the ship
position values in the structure variable
                    b.battleship[counter][1] = startPositionCol;
                    counter++;   // increase the counter one
                } //end  for
                break;
            } //end else

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

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

        }

        if (tolower(shipOrientiation) == 'h')  { // check for horizontal setup
            int startPositionRow = letterToRowNumber(shipPosition[0]); // Get the start
value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';
            bool isConflict = conflictWithOtherShip(playerMatrix,
                  startPositionRow, startPositionCol, DESTROYER_LENGTH,
                    tolower(shipOrientiation)); // check the ship is conflict with
other ship position
            if ((startPositionCol > DESTROYER_LENGTH + 3) || isConflict) { // not able
to setup the battleship horizontally from this position
                cout << "You cannot place the battleship in this position."
                        " TRY AGAIN!" << endl;
                shipPosition.clear();
                continue;
            } //end if
            else
            {
                int counter = 0;
                auto rowIt = std::next(playerMatrix.begin(), startPositionRow); //
Accessing the row

                for (int i = startPositionCol; i < startPositionCol
                        + DESTROYER_LENGTH; i++)
                {
                    auto colIt = std::next(rowIt->begin(), i);
                    *colIt = 'D';    // set the player matrix with 'A' for indicating
the aircraft location
                    //playerMatrix[startPositionRow][i] = 'D';
// set the player matrix with 'D' for indicating the battle location
                    b.destroyer[counter][0] = startPositionRow;
// insert the ship position values in the structure variable
                    b.destroyer[counter][1] = i;
                    counter++;                                                    //
increase the counter one
                }
                break;
            } //end else
        }
        if (tolower(shipOrientiation) == 'v')                                    //
check for the vertical setup
        {
            int startPositionRow = letterToRowNumber(shipPosition[0]);      // Get the
start value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';                          //
make a character value to integer
            bool isConflict = conflictWithOtherShip(playerMatrix,
                    startPositionRow, startPositionCol, DESTROYER_LENGTH,
                    tolower(shipOrientiation)); // check the ship is conflict with
other ship position
            if ((startPositionRow > DESTROYER_LENGTH + 3) || 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;
            }
            else
            {
                int counter = 0;
                for (int i = startPositionRow; i < startPositionRow
                        + DESTROYER_LENGTH; i++)
                {
                    auto rowIt = std::next(playerMatrix.begin(), i);
                    auto colIt = std::next(rowIt->begin(), startPositionCol); //
Accessing the column
                    *colIt = 'D';
                    //playerMatrix[i][startPositionCol] = 'D'; // set the player matrix
with 'D' for indicating the battle location
                    b.destroyer[counter][0] = i;             // insert the ship
position values in the structure variable
                    b.destroyer[counter][1] = startPositionCol;
                    counter++;// increase the counter one
                }
                break;
            }

        }
    }

    drawPlayerArea(playerMatrix); // 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'
            if (tolower(shipOrientiation) == 'h' ||
                    tolower(shipOrientiation) == 'v') {    // compare the ship
orientation input if it is 'v' or 'h' then fine
                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 corvette position without a space "
                "(example: a0, a1...): " << endl;
        while (true) {  // take a infinite loop for satisfying the valid input for
corvette 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
                }
                else {
                    cout << "Enter a valid corvette position without a space "
                            "(example: a0, a1...): " << endl;   // ask for valid input
again
                    shipPosition.clear();
                    continue;
                }
            }
            else {
                cout << "Enter a valid corvette position without a space"
                        " (example: a0, a1...): " << endl;   // ask for valid input
again
                shipPosition.clear();
                continue;
            }

        }

        if (tolower(shipOrientiation) == 'h')  // check for horizontal setup
        {
            int startPositionRow = letterToRowNumber(shipPosition[0]);      // Get the
start value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';
            bool isConflict = conflictWithOtherShip(playerMatrix,
                 startPositionRow, startPositionCol, CORVETTE_LENGTH,
                 tolower(shipOrientiation)); // check the ship is conflict with other
ship position
            if ((startPositionCol > CORVETTE_LENGTH + 4) || isConflict) {  // not able
to setup the battleship horizontally from this position
                cout << "You cannot place the corvette in this position. TRY AGAIN!" <<
endl;
                shipPosition.clear();
                continue;
            }
            else
            {
                int counter = 0;
                auto rowIt = std::next(playerMatrix.begin(), startPositionRow); //
Accessing the row
                for (int i = startPositionCol; i < startPositionCol
                        + CORVETTE_LENGTH; i++)
                {
                    auto colIt = std::next(rowIt->begin(), i);
                    *colIt = 'C';    // set the player matrix with 'A' for indicating
the aircraft location

                     //playerMatrix[startPositionRow][i] = 'C';  // set the player
matrix with 'C' for indicating the battle location
                    b.corvette[counter][0] = startPositionRow;  // insert the ship
position values in the structure variable
                    b.corvette[counter][1] = i;
                    counter++; // increase the counter one
```

```
                }
                break;
            }
        }
        if (tolower(shipOrientiation) == 'v') { // check for the vertical setup
            int startPositionRow = letterToRowNumber(shipPosition[0]);      // Get the
start value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
            int startPositionCol = shipPosition[1] - '0';                          //
make a character value to integer
            bool isConflict = conflictWithOtherShip(playerMatrix,
                    startPositionRow, startPositionCol, CORVETTE_LENGTH,
                    tolower(shipOrientiation)); // check the ship is conflict with
other ship position
            if ((startPositionRow > CORVETTE_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;
            }
            else
            {
                int counter = 0;
                for (int i = startPositionRow; i < startPositionRow + CORVETTE_LENGTH;
i++)
                {
                    auto rowIt = std::next(playerMatrix.begin(), i);
                    auto colIt = std::next(rowIt->begin(), startPositionCol); //
Accessing the column
                    *colIt = 'C';
                    //playerMatrix[i][startPositionCol] = 'C';// set the player matrix
with 'C' for indicating the battle location
                    b.corvette[counter][0] = i;  // insert the ship position values in
the structure variable
                    b.corvette[counter][1] = startPositionCol;
                    counter++;  // increase the counter one
                }
                break;
            }

        }
    } //end while-loop
    drawPlayerArea(playerMatrix); // redraw the player area with the position of
destroyer
    return b;
} //end of setYourBattleShip

//*************************************************************
//Definition of function setComputerBattleShip                    *
//Set computer ship position and return ship position             *
//*************************************************************
BShipSetUp BShipSetUp::setComputerBattleShip(BShipSetUp computerShipPosition) {
    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
    computerMatrix = getMatrixData();
    //BShipSetUp computerShipPosition = BShipSetUp();  // Declare a structure variable
to store computer ship position
    //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;
    {
        auto rowIt = std::next(computerMatrix.begin(), rowPosition); // Accessing the
row
        int counter = 0;
        for (int i = colPosition; i < colPosition + AIRCRAFT_LENGTH; i++)
        {
            auto colIt = std::next(rowIt->begin(), i); //assigning data using iterator
            *colIt = 'A';    // set the player 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
        }
    }
    else
    {                                                                       //
Otherwise orientation is vertical
        int counter = 0;
        for (int i = rowPosition; i < rowPosition + AIRCRAFT_LENGTH; i++)
        {
            auto rowIt = std::next(computerMatrix.begin(), i);
            auto colIt = std::next(rowIt->begin(), colPosition); // Accessing the
column
            *colIt = 'A';
            computerShipPosition.aircraft[counter][0] = i;              // Insert the
ship position values in the structure variable
            computerShipPosition.aircraft[counter][1] = colPosition;
            counter++;                                                  // Increase the
counter one
        }
    }

    //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;
        auto rowIt = std::next(computerMatrix.begin(), rowPosition); // Accessing the
row
        for (int i = colPosition; i < colPosition + BATTLESHIP_LENGTH; i++)
        {
            auto colIt = std::next(rowIt->begin(), i);
```

```
            *colIt = 'B';    // set the player matrix with 'B' for indicating the
aircraft location
            //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
        }
    }
    else
    {                                                                        //
Otherwise orientation is vertical
        int counter = 0;
        for (int i = rowPosition; i < rowPosition + BATTLESHIP_LENGTH; i++)
        {
            auto rowIt = std::next(computerMatrix.begin(), i);
            auto colIt = std::next(rowIt->begin(), colPosition); // Accessing the
column
            *colIt = 'B';
            //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 corvette
    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;
        auto rowIt = std::next(computerMatrix.begin(), rowPosition); // Accessing the
row

        for (int i = colPosition; i < colPosition + DESTROYER_LENGTH; i++)
        {
            auto colIt = std::next(rowIt->begin(), i);
            *colIt = 'D';    // set the player matrix with 'D' for indicating the
aircraft location
            //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
        }
    }
    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++)
        {
            auto rowIt = std::next(computerMatrix.begin(), i);
            auto colIt = std::next(rowIt->begin(), colPosition); // Accessing the
column
            *colIt = 'D';
            //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
        }
    }
    //setup the destroyer
    shipOrientation = (rand() % 2);                    // Randomly select ship
orientation for destroyer
    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;
        auto rowIt = std::next(computerMatrix.begin(), rowPosition); // Accessing the
row
        for (int i = colPosition; i < colPosition + CORVETTE_LENGTH; i++)
        {
            auto colIt = std::next(rowIt->begin(), i);
            *colIt = 'C';    // set the player matrix with 'D' for indicating the
aircraft location
            //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
        }
    }
    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++)
        {
            auto rowIt = std::next(computerMatrix.begin(), i);
```

```
            auto colIt = std::next(rowIt->begin(), colPosition); // Accessing the
column
            *colIt = 'C';
            //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
        }
    }
    return computerShipPosition;
}
//*************************************************************
//Definition of function inorderTraversal
//This function used to traversal of tree nodes
//*************************************************************
void BShipSetUp::inorderTraversal(TreeNode* root, list<char>& nodes) {
    // If the current root is null, return (base case of recursion).
    if (!root) return;

    // Recursively traverse the left subtree to visit all nodes in the left branch.
    inorderTraversal(root->left, nodes);

    // Process the current root node by adding its value to the list.
    nodes.push_back(root->value);

    // Recursively traverse the right subtree to visit all nodes in the right branch.
    inorderTraversal(root->right, nodes);
}
//*************************************************************
//Definition of function insert
//This function used to insert a tree node
//*************************************************************
BShipSetUp::TreeNode* BShipSetUp::insert(TreeNode* root, char value) {
    //  If the current root is null (empty tree or correct position found),
    //         create a new TreeNode with the given value and return it.
    if (!root)
        return new TreeNode(value);

    //  If the value to be inserted is smaller than the current root's value,
    //         recursively insert the value into the left subtree.
    if (value < root->value)
        root->left = insert(root->left, value);

    //  If the value to be inserted is greater than or equal to the current root's
value,
    //         recursively insert the value into the right subtree.
    else
        root->right = insert(root->right, value);

    // Step 4: Return the root node after insertion (unchanged for existing nodes).
    return root;
}
//*************************************************************
//Definition of function showPlayZone
//This draw the computer and player play zone
//*************************************************************
void BShipSetUp::showPlayZone() {
    // Create the binary tree for y-coordinates
```

```
    TreeNode* yCoordRoot = nullptr;
    for (char c = 'A'; c <= 'J'; ++c) {
        yCoordRoot = insert(yCoordRoot, c);
    }

    // Get the in-order traversal of the tree
    list<char> yCoordList;
    inorderTraversal(yCoordRoot, yCoordList);

    cout << " -          Computer Zone                          Your Zone      -" <<
endl;
    cout << " - - - 0 1 2 3 4 5 6 7 8 9 -          - - - 0 1 2 3 4 5 6 7 8 9 -" <<
endl;

    auto yItem = yCoordList.begin();  // Iterator for y-coordinates
    auto playerRow_it = playerMatrix.begin(); // Iterator for playerMatrix

    for (auto row_it = computerMatrix.begin(); row_it != computerMatrix.end();
++row_it) {
        cout << " | " << *yItem << " | ";  // Access the character value from the list
        for (auto elem_it = row_it->begin(); elem_it != row_it->end(); ++elem_it) {
            if (*elem_it >= 'A' && *elem_it <= 'D') // Hide the computer ship position
                cout << "* ";   // Display the computer matrix
            else
                cout << *elem_it << " ";   // Display the computer matrix
        }

        cout << "|           | " << *yItem << " | ";  // Set the y-column 'A' to 'J'
        for (auto ele_it = playerRow_it->begin(); ele_it != playerRow_it->end();
++ele_it) {
            cout << *ele_it << " ";   // Display the player matrix
        }
        cout << "|" << endl;

        ++yItem;           // Move to the next element in the list
        ++playerRow_it;    // Move to the next row in the playerMatrix
    }
}

//*************************************************************
//Definition of function startPlay                    *
//This function allow to input player and computer attack position each other   *
//*************************************************************
void BShipSetUp::startPlay(BShipSetUp battleShipPlayer, BShipSetUp battleShipComputer)
{
    // Create and initialize the graph
    Graph graph;
    graph.initialize();

    // Define edges (optional; for adjacency logic)
    for (int i = 0; i < Graph::MAX_NODES - 1; ++i) {
        graph.addEdge(i, i + 1); // Example: Connect sequential nodes
    }

    // Find min and max labels
    pair<char, char> minMaxLabels = graph.findMinMaxLabels();
    char minLabel = minMaxLabels.first;
    char maxLabel = minMaxLabels.second;

    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;
    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...): " << endl;
            getline(cin, attackPosition);                     // Get the ship position
            if (attackPosition.length() == POSITION_LENGTH) { // position length should
the 2 character length
                for (auto& c : attackPosition) c = toupper(c);   // make the uppercase
of the input position for comparing value and allow for lower or upper case character
                if ((attackPosition[0] >= minLabel && attackPosition[0] <= maxLabel)
                    && (attackPosition[1] >= '0' && attackPosition[1] <= '9')) { //
check for valid input and convert number to character
                    rowPosition = letterToRowNumber(attackPosition[0]);      // Get the
start value of y-axis (0,1,2...9) from the letter position (A,B,C....J)
                    colPosition = attackPosition[1] - '0';                   // Convert
the column position character to integer
                    isSuccessful = checkSuccessful(battleShipComputer, 'p',
                        rowPosition, colPosition); // Check it is successfully hit or
not
                    if (isSuccessful) {                         // If successfully hit
then player will get another change for attack
                        trackWin = displayShipStatus(battleShipPlayer,
                            battleShipComputer);
                        gameHead();                   // Draw the game head
                        showPlayZone();   // Draw the play zone
                        cout << " You attack successfully !!! " << endl;
                        attackPosition.clear();
                        if (trackWin == 0) break; // Player won the game; do not need
continue
                        continue;
                    }
                    else {
                        trackWin = displayShipStatus(battleShipPlayer,
                            battleShipComputer);
                        gameHead();                   // Draw the game head
                        showPlayZone();   // 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
                    }
                }
                else {
                    cout << "----Enter a valid aircraft position without a "
                        "space (example: a0, a1...)----" << endl;   // ask for valid
input again
                    attackPosition.clear();
                    continue;
                }
            }
            else {
                cout << "----Enter a valid aircraft position without a space"
                    " (example: a0, a1...)----" << endl;   // ask for valid input again
                attackPosition.clear();
                continue;
```

```
            }
        }
        if (trackWin == 0)  // The player won
        {
            cout << "Congratulation!!! ~~~" << playerName << "~~~ "
                "You won this game!!!" << endl;
            break;
        }
        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(battleShipPlayer, 'c',
                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(battleShipPlayer, battleShipComputer);
                gameHead();                    // Draw the game head
                showPlayZone();   // Draw the play zone
                cout << "Computer attack the position: " << graph.getLabel(rowPosition)
                    << colPosition << endl;
                cout << "Computer attack your ship successfully !!! " << endl;
                if (trackWin == 1) break; // Computer won the game; do not need
continue
                continue;
            }
            else {
                trackWin = displayShipStatus(battleShipPlayer, battleShipComputer);
                gameHead();                    // Draw the game head
                showPlayZone();   // Draw the play zone
                cout << "Computer attack the position: " << graph.getLabel(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
            }
        }
        if (trackWin == 1)  // The computer won
        {
            cout << "Congratulation!!! ~~~ Computer ~~~ won this game!!!\n";
            break;
        }

    }
}
//*************************************************************
//Definition of function checkSuccessful                       *
//This function check the attack is successful or not and      *
//Update the ship position structure. Return boolean status    *
*
//*************************************************************
bool BShipSetUp::checkSuccessful(BShipSetUp& bShipInfo, char sourceData,
    int row, int col) {
    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;
    for (int i = 0; i < AIRCRAFT_LENGTH; i++) {         // Check the air craft position
        if (bShipInfo.aircraft[i][0] != -1)         // If the position not hit yet
        {
            if (bShipInfo.aircraft[i][0] == row && bShipInfo.aircraft[i][1] == col)
//hit successful
            {
                bShipInfo.aircraft[i][0] = -1;   // Track this position destroy
                if (sourceData == 'c')
                {
                    // Iterate to the correct row
                    auto rowIt = playerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)
                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully
                }
                else
                {
                    // Iterate to the correct row
                    auto rowIt = computerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)

                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully

                    //computerMatrix[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 (bShipInfo.battleship[i][0] != -1)   // If the position not hit yet
        {
            if (bShipInfo.battleship[i][0] == row &&
                bShipInfo.battleship[i][1] == col) //hit successful
            {
                bShipInfo.battleship[i][0] = -1;   // Track this position destroy
                if (sourceData == 'c')
                {
                    // Iterate to the correct row
                    auto rowIt = playerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)
                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully
                    //playerMatrix[row][col] = '@';        // Set the matrix position
'@' if hit successfully
                }
```

```
                else
                {
                    // Iterate to the correct row
                    auto rowIt = computerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)

                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully

                    //computerMatrix[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 (bShipInfo.destroyer[i][0] != -1)   // If position not hit yet
        {
            if (bShipInfo.destroyer[i][0] == row &&
                bShipInfo.destroyer[i][1] == col) //hit successful
            {
                bShipInfo.destroyer[i][0] = -1;   // Track this position destroy
                if (sourceData == 'c')
                {
                    // Iterate to the correct row
                    auto rowIt = playerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)
                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully
                    //playerMatrix[row][col] = '@';        // Set the matrix position
'@' if hit successfully
                }
                else
                {
                    // Iterate to the correct row
                    auto rowIt = computerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)

                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully

                    //computerMatrix[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 (bShipInfo.corvette[i][0] != -1)   // If position not hit yet
        {
            if (bShipInfo.corvette[i][0] == row &&
                bShipInfo.corvette[i][1] == col) //hit successful
            {
                bShipInfo.corvette[i][0] = -1;   // Track this position destroy
                if (sourceData == 'c')
                { // Iterate to the correct row
                    auto rowIt = playerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)
                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully
                    //playerMatrix[row][col] = '@'; // Set the matrix position '@' if
hit successfully
                }
                else
                {
                    // Iterate to the correct row
                    auto rowIt = computerMatrix.begin();
                    advance(rowIt, row);  // Move iterator to the 2nd row (1-based)

                    // Iterate to the correct column within the row
                    auto colIt = rowIt->begin();
                    advance(colIt, col);  // Move iterator to the 3rd column (1-based)
                    *colIt = '@';       // Set the matrix position '@' if hit
successfully
                    //computerMatrix[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
    if (sourceData == 'c')
    {
        // Iterate to the correct row
        auto rowIt = playerMatrix.begin();
        advance(rowIt, row);  // Move iterator to the 2nd row (1-based)
        // Iterate to the correct column within the row
        auto colIt = rowIt->begin();
        advance(colIt, col);  // Move iterator to the 3rd column (1-based)
        *colIt = 'o';       // Set the matrix position '@' if hit successfully
        //playerMatrix[row][col] = 'o';  // Set the matrix position 'o' if hit
unsuccessful
    }
    else
    {
        // Iterate to the correct row
        auto rowIt = computerMatrix.begin();
        advance(rowIt, row);  // Move iterator to the 2nd row (1-based)

        // Iterate to the correct column within the row
        auto colIt = rowIt->begin();
```

```
        advance(colIt, col);  // Move iterator to the 3rd column (1-based)
        *colIt = 'o';       // Set the matrix position 'o' if hit unsuccessful
        //computerMatrix[row][col] = 'o'; // Set the matrix position '@' if hit
successfully
    }
    return isFound;
}
//*************************************************************
//Definition of function displayShipStatus
//Check the ship status using structure reference
//Draw the ship status
//Return the integer indicating the game is over or not
//*************************************************************
int BShipSetUp::displayShipStatus(BShipSetUp player, BShipSetUp computer)
{
    system("cls");
    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

    bool isFound = false;
    //Count aircraft
    for (int i = 0; i < AIRCRAFT_LENGTH; i++) // Check the air craft position
    {
        if (computer.aircraft[i][0] != -1)   // If the position not hit yet
        {
            airCarftComputer += 1;           // Count the computer air craft
        }
        if (player.aircraft[i][0] != -1)
        {
            airCarftPlayer += 1;            // Count the player air craft
        }
    }
    //Count battleship
    for (int i = 0; i < BATTLESHIP_LENGTH; i++) // Check the battle ship position
    {
        if (computer.battleship[i][0] != -1)    // If the position not hit yet
        {
            battelShipComputer += 1;           // Count the computer battle ship
        }
        if (player.battleship[i][0] != -1)
        {
            battelShipPlayer += 1;        // Count the player battle ship
        }
    }
    //Count destroyer
    for (int i = 0; i < DESTROYER_LENGTH; i++)  // Check the Destroyer position
    {
        if (computer.destroyer[i][0] != -1)     // If the position not hit yet
        {
            destroyerComputer += 1;         // Count the computer destroyer ship
        }
```

```
            if (player.destroyer[i][0] != -1)
            {
                destroyerPlayer += 1;            // Count the player destroyer ship
            }
        }
        //Count corvette
        for (int i = 0; i < CORVETTE_LENGTH; i++)     // Check the corvette position
        {
            if (computer.corvette[i][0] != -1)        // If the position not hit yet
            {
                corvetteComputer += 1;           // Count the computer corvette ship
            }
            if (player.corvette[i][0] != -1)
            {
                corvettePlayer += 1;           // Count the player corvette ship
            }
        }
        cout << "....................................................."
            ".........\n";
        cout << " -  Computer Ships Status                Your Ship Status-\n";
    //Display the ship status
        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: " << corvetteComputer << " units                "
            " | ----Corvette: " << corvettePlayer << " units" << endl;
        cout << "....................................................."
            ".........\n";
        // Check all computer ships were hit successfully
        if (airCarftComputer == 0 && battelShipComputer == 0 &&
            destroyerComputer == 0 && corvetteComputer == 0)
        {
            return 0;   // Computer loss and player win
        }
        // Check all player ships were hit successfully
        else if (airCarftPlayer == 0 && battelShipPlayer == 0 &&
            destroyerPlayer == 0 && corvettePlayer == 0)
        {
            return 1;   // player loss and computer win
        }
        else
        {
            return 2; // Continue play
        }
} //end of displayShipStatus function


/*
 * File: BattleShip.cpp
 * Author: Khadiza Akter
 * Created on December 02, 2024, 10:02 PM
 * Purpose: Implementation file for Graph class
 */
#include <iostream>
#include <array>
#include <string>
```

```cpp
#include <algorithm>
#include <cstdlib>

using namespace std;
#include "Graph.h"

// Initialize the graph with labels
void Graph::initialize() {
    for (int i = 0; i < MAX_NODES; ++i) {
        nodes[i].label = 'A' + i; // Assign labels 'A' to 'J'
    }
}

// Add an edge between two nodes
void Graph::addEdge(int u, int v) {
    if (u >= 0 && u < MAX_NODES && v >= 0 && v < MAX_NODES) {
        nodes[u].edges[v] = 1; // Mark as connected
        nodes[v].edges[u] = 1; // For undirected graph
    }
}

// Get the label of a node
char Graph::getLabel(int node) const {
    if (node >= 0 && node < MAX_NODES) {
        return nodes[node].label;
    }
    return '?'; // Return '?' for invalid node
}

// Find the minimum and maximum labels
pair<char, char> Graph::findMinMaxLabels() const {
    char minLabel = nodes[0].label;
    char maxLabel = nodes[0].label;

    for (int i = 1; i < MAX_NODES; ++i) {
        if (nodes[i].label < minLabel) {
            minLabel = nodes[i].label;
        }
        if (nodes[i].label > maxLabel) {
            maxLabel = nodes[i].label;
        }
    }
    return { minLabel, maxLabel };
}
```

```cpp
/*
 * File: BattleShip.h
 * Author: Khadiza Akter
 * Created on October 07, 2024, 10:24 PM
 * Purpose: Specification file for BattleShip class
 */
#ifndef BATTLESHIP_H
#define BATTLESHIP_H
```

```
#include <list> // Include list for STL list
//using namespace std;
class BattleShip {
private:
    std::list<std::list<char>> listMatrix; // Use std::list<std::list<char>>

public:
    const int ROWS = 10; // Constant rows for 10x10 matrix
    const int COLS = 10; // Constant columns for 10x10 matrix

    // Constructor fills the arrMatrix with default values
    BattleShip() {
        listMatrix = list<list<char>>(ROWS, list<char>(COLS, '*'));
    }

    // Accessor inline function, return filled matrix
    list<list<char>> getMatrixData() const {
        return listMatrix;
    }

};

#endif /* BATTLESHIP_H */


 /*
  * File:   BShipSetUp.h
  * Author: Khadiza Akter
  * Created on November 21, 2022, 1:50 PM
  * Purpose: Specification file for derived class BShipSetUp
  */

#ifndef BSHIPSETUP_H
#define BSHIPSETUP_H

#include <iostream> //Input-output library
#include <cstdlib>  //Srand to set the seed or system()
#include <string>
#include <vector>
using namespace std;//Standard Name-space under which System Libraries reside

#include "BattleShip.h" //needed for base class

//This a derived class from base class BattleShip
class BShipSetUp:public BattleShip {    // Declare a class for different types battle
ship
private:
    //static char yCoord[11]; // declare a character array for maintain y-coordinate as a
character
    string playerName; // Declare the input array to take player name
    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 its coordinate values (row,col)
    int corvette[2][2]; // Corvette length is 2 for tracking its coordinate values
(row,col)
    //Check the ship position conflict with other ship or not
    bool conflictWithOtherShip(list<std::list<char>>, int, int, int, char);
```

```
public:
    //Exception class for InvalidName
    class InvalidName {};
    //constructor #1
    BShipSetUp() {
        playerName = "";
    }
    //constructor #2
    BShipSetUp(string name) {
        bool space = false;
        if (name.empty()) throw InvalidName();
        else if (name.size() > 0) {
            for (int i = 0; i < name.size(); i++) {
                if (isspace(name[i])) space = true;
                else {
                    space = false;
                    break;
                }
            }
            if (space == true) throw InvalidName();
            else               playerName = name;
        }
    }
    // Tree node structure
    struct TreeNode {
        char value;
        TreeNode* left;
        TreeNode* right;
        TreeNode(char val) : value(val), left(nullptr), right(nullptr) {}
    };
    void inorderTraversal(TreeNode*, list<char>&);
    TreeNode* insert(TreeNode*, char);
    int shipSizes[4] = { 2, 3, 4, 5}; // ship sizes
    list<list<char>> playerMatrix; // to hold player matrix information
    list<list<char>> computerMatrix;// to hold computer Matrix information
    string getPlayerName()const { return playerName; }
    // mutator function,and set player ship position and return ship members
    BShipSetUp setYourBattleShip(BShipSetUp);
    //Set computer ship position and return ship members
    BShipSetUp setComputerBattleShip(BShipSetUp);
    //Determine the letter (A,...J) value to integer y-axis
    //value (0,1,2...9)
    friend int letterToRowNumber(char);
    void drawPlayerArea(list<std::list<char>>); //Display player matrix
    void drawRow(list<list<char>>::iterator, list<list<char>>::iterator, char[], int);//
Function to draw rows of the grid recursively
    void drawElements(list<char>::iterator, list<char>::iterator);//Function to draw
elements of a row recursively
    void gameHead(); //Display player name
    //operator= overloaded
    const BShipSetUp operator=(const BShipSetUp& right);
    void showPlayZone(); // Draw the computer and player zone
    void startPlay(BShipSetUp, BShipSetUp); // Allow the player and computer for attack
the ships
    bool checkSuccessful(BShipSetUp& bShipInfo, char sourceData,
        int row, int col);// Check where the attack is successful or not
    int displayShipStatus(BShipSetUp, BShipSetUp);
    void quickSort(int[], int, int);
```

```
};

#endif /* BSHIPSETUP_H */


#pragma once
/*
 * File:   BShipSetUp.h
 * Author: Khadiza Akter
 * Created on December 02, 2024, 1:50 PM
 * Purpose: Create a graph for the label column
 */

#ifndef GRAPH_H
#define GRAPH_H
#include <iostream>
#include <array>
#include <string>
#include <algorithm>
#include <cstdlib>

using namespace std;


class Graph {
public:
    // Define a structure for a graph node
    struct Node {
        char label;          // Label of the node (e.g., 'A', 'B', ...)
        int edges[10];       // Array to represent edges to other nodes (1: connected, 0:
not connected)

        Node() : label(' '), edges{ 0 } {} // Default constructor
    };
    static const int MAX_NODES = 10; // Maximum number of nodes
    Node nodes[MAX_NODES];           // Array of nodes
    void initialize();
    void addEdge(int u, int v);
    char getLabel(int node) const;
    pair<char, char> findMinMaxLabels() const;
};
#endif /* GRAPH_H */
```