Hacettepe University

Department of Computer Engineering

BBM103 Assignment 4 Report

Sudenaz Yazıcı – 2210356008

02.01.2023

Analysis

In this problem, we are asked to code the Battleship game. Two players plays this game on 10x10 squared grid. Each player has hidden board that they adjust the locations of their ships. Also, each player has another board that they keep track of their moves and if they sinked the opponent’s ship or not. There are five different class of ships that have different lenght.

Before the game begins, players determine their ships’ locations on their hidden boards. After that, players get to make their move in turns.

If a ship’s all squares completely sinked, the program shows that to players. Moreover, if one player’s all ships sinks, the game finishes and the program shows the winner. Additionally, at the end, the program shows the hidden missed squares of the players’ boards.

Design

Importing Modules and Arranging Lists/Variables

Firstly, I imported os, sys and defined some empty lists, variables that will be used later in the program.

Reading Files

The program reads total of six files which two of them are fixed and four of them are taken from the command line from the user.

The program reads files one by one and records informations using lists and variables I defined at the beginning. In addition, IOError is checked for every file and if it occurs the program shows which files cannot be opened. Also, when reading the last file, we create game boards for both players and we define a function that returns the letter’s index.

Writing to File

We open the file which we will write into and all other processes take place.

Inside of this part, if an error occurs while reading files, the program prints files which cannot be opened.

Showing the Grids Function

This function basically prints two players’ boards and situation of ships after each move.

Controlling the Ships Function

I defined two separate controlling functions for two players.

Player’s Functions

I defined two separate functions for each player. I will briefly explain what the program does inside these functions.

First of all, it reads one move at a time and checks the given position. If player misses ship, it changes the information on the board to “O”. Otherwise, the position on the board becomes “X”. When checking the positions, it also checks the ship classes.

If an error does not ocur, one player’s function calls the other player’s function (mutual recursion). If there is an error, it will print the suitable message and calls itself.

The program will continue until the moves finishes. At the end, it prints final informations.

Programmer’s Catalogue

Time Spent for Analysis: Analysis section includes reading and understanding the problem. Since the problem is a game I know, this section took me about half an hour.

Time Spent for Design: Design section includes writing a pseudocode and designing an algorithm. This section took me about one and half an hour.

Time Spent for Implementation: Implementation section took more time than other sections since I tried to learn some new things and I ran into some errors. I figured out how to use recursion. Then, I coded everything according to my pseudocode. Overall, I spent 15 hours on this section.

Time Spent for Testing: Since the possible situations we must control are given, this part did not take long time. Overall I can say I spent an hour on this section.

Time Spent for Reporting: Reporting section includes writing and explaining the problem and the program. I looked up some examples before writing the report. This section took me about 5 hours.

#The Program

```
import os
import sys

error_file = []  # if any error occurs while trying to open files, they
will be added to this list
p1_hidden_board = []
p2_hidden_board = []
p1_moves = []
p2_moves = []
p1_board = []
p2_board = []
p1_ships = ["-", "- -", "-", "-", "- - - -"]
p2_ships = ["-", "- -", "-", "-", "- - - -"]
# sinked ship number of player1
ship_c1, ship_b1, ship_d1, ship_s1, ship_p1 = 0, 0, 0, 0, 0
```

```
# sinked ship number of player2
ship_c2, ship_b2, ship_d2, ship_s2, ship_p2 = 0, 0, 0, 0, 0
current_dir_path = os.getcwd()

reading_file_name = "OptionalPlayer1.txt"
reading_file_path = os.path.join(current_dir_path, reading_file_name)
with open(reading_file_path, "r") as f:
    optional_p1 = f.read().split("\n")
    for i in range(len(optional_p1)):
        optional_p1[i] = optional_p1[i].replace(":",";")
        optional_p1[i] = optional_p1[i].split(";")
reading_file_name = "OptionalPlayer2.txt"
reading_file_path = os.path.join(current_dir_path, reading_file_name)
with open(reading_file_path, "r") as f:
    optional_p2 = f.read().split("\n")
    for i in range(len(optional_p2)):
        optional_p2[i] = optional_p2[i].replace(":",";")
        optional_p2[i] = optional_p2[i].split(";")
try:
    reading_file_name = sys.argv[1]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.readline().split("\n")
        while line_data != [""]:  # the program reads input file line by
line until the empty line
            if len(line_data) == 2:
                line_data.remove(line_data[1])
            line_data[0] = line_data[0].split(";")
            p1_hidden_board.extend(line_data)
            line_data = f.readline().split("\n")
except IOError:
    error_file.append(reading_file_name)

try:
    reading_file_name = sys.argv[2]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.readline().split("\n")
        while line_data != [""]:  # the program reads input file line by
line until the empty line
            if len(line_data) == 2:
                line_data.remove(line_data[1])
            line_data[0] = line_data[0].split(";")
            p2_hidden_board.extend(line_data)
            line_data = f.readline().split("\n")
except IOError:
    error_file.append(reading_file_name)

try:
    reading_file_name = sys.argv[3]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.read()
        line_data = line_data.split(";")
        p1_moves.extend(line_data)
except IOError:
    error_file.append(reading_file_name)

try:
    reading_file_name = sys.argv[4]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.read()
        line_data = line_data.split(";")
        p2_moves.extend(line_data)

        # with for loop down below, we create the game board for both
players
        for i in range(10):
            row = []
            for j in range(10):
                row.append("-")
            p1_board.append(row)
        for i in range(10):  # I wrote separate for loops because
            row = []  # if I write them in the same loop they would
represent the same list
            for j in range(10):
                row.append("-")
            p2_board.append(row)

        p1_moves = p1_moves[:-1]
        p2_moves = p2_moves[:-1]

        letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]

        def search_letters(move):  # function that returns the index of
letters
            for letter in letters:
                if letter == move:
                    return letters.index(letter)
except IOError:
    error_file.append(reading_file_name)

writing_file_name = "Battleship.out"
writing_file_path = os.path.join(current_dir_path, writing_file_name)
with open(writing_file_path, "w") as f:
    if len(error_file) != 0:  # if at least one file cannot be opened,
program writes IOError
        names = ""
        for name in error_file:
            names = names + name + ","
        names = names[:-1]  # removing last comma at the end
        print("IOError: input file(s) {} is/are not
reachable.".format(names))
        f.write("IOError: input file(s) {} is/are not
reachable.\n".format(names))
        exit()
    print("Battle of Ships Game\n")
    f.write("Battle of Ships Game\n\n")
    move_num1 = 0
    move_num2 = 0
    round_num = 1

    def show_grid():
        for i in range(10):
            print(i + 1, end=" ")
            f.write(str(i+1)+" ")
            for j in range(10):
                print(p1_board[i][j], end=" ")
                f.write(p1_board[i][j]+" ")
            print("\t" + str(i + 1), end=" ")
            f.write("\t"+str(i+1)+" ")
            for j in range(10):
                print(p2_board[i][j], end=" ")
                f.write(p2_board[i][j]+" ")
            print()
            f.write("\n")
        print()
        print("Carrier\t\t"+p1_ships[0]+"\tCarrier\t\t"+p2_ships[0])
        print("Battleship\t" + p1_ships[1] + "\tBattleship\t" +
p2_ships[1])
        print("Destroyer\t" + p1_ships[2] + "\tDestroyer\t" + p2_ships[2])
        print("Submarine\t" + p1_ships[3] + "\tSubmarine\t" + p2_ships[3])
        print("Patrol Boat\t" + p1_ships[4] + "\tPatrol Boat\t" +
p2_ships[4])
        print()
        f.write("\n")
        f.write("Carrier\t\t" + p1_ships[0] + "\t\t\tCarrier\t\t" +
p2_ships[0]+"\n")
        f.write("Battleship\t" + p1_ships[1] + "\t\t\tBattleship\t" +
p2_ships[1]+"\n")
        f.write("Destroyer\t" + p1_ships[2] + "\t\t\tDestroyer\t" +
p2_ships[2]+"\n")
        f.write("Submarine\t" + p1_ships[3] + "\t\t\tSubmarine\t" +
p2_ships[3]+"\n")
        f.write("Patrol Boat\t" + p1_ships[4] + "\t\tPatrol Boat\t" +
p2_ships[4]+"\n")
        f.write("\n")

    def control_ships_p1():  # shows sinked ships based on sinked boxes
        if ship_c2 == 5:
            p2_ships[0] = "X"
        if ship_b2 == 1:
            p2_ships[1] = "X -"
        elif ship_b2 == 2:
            p2_ships[1] = "X X"
        if ship_d2 == 3:
            p2_ships[2] = "X"
        if ship_s2 == 3:
            p2_ships[3] = "X"
        if ship_p2 == 1:
            p2_ships[4] = "X - - -"
        elif ship_p2 == 2:
            p2_ships[4] = "X X - -"
        elif ship_p2 == 3:
            p2_ships[4] = "X X X -"
        elif ship_p2 == 4:
            p2_ships[4] = "X X X X"
    def control_ships_p2():
        if ship_c1 == 5:
            p1_ships[0] = "X"
        if ship_b1 == 1:
            p1_ships[1] = "X -"
        elif ship_b1 == 2:
            p1_ships[1] = "X X"
        if ship_d1 == 3:
            p1_ships[2] = "X"
        if ship_s1 == 3:
            p1_ships[3] = "X"
        if ship_p1 == 1:
            p1_ships[4] = "X - - -"
        elif ship_p1 == 2:
            p1_ships[4] = "X X - -"
        elif ship_p1 == 3:
            p1_ships[4] = "X X X -"
        elif ship_p1 == 4:
            p1_ships[4] = "X X X X"
    def player1():
        global round_num, move_num1, p2_ships, ship_c2, ship_b2, ship_d2,
ship_s2, ship_p2
        while move_num1 < min(len(p1_moves), len(p2_moves)):
            try:
                print("Player1's Move\n")
                print("Round : {}".format(round_num) + "\t\t" + "Grid Size:
10x10\n")
                print("Player1's Hidden Board" + "\t" + "Player2's Hidden
Board")
                print("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I
J")
                f.write("Player1's Move" + "\n\n")
                f.write("Round : {}".format(round_num) + "\t\t\t\t" + "Grid
Size: 10x10" + "\n\n")
                f.write("Player1's Hidden Board" + "\t" + "Player2's Hidden
Board" + "\n")
                f.write("  A B C D E F G H I J" + "\t" + "  A B C D E F G H
I J" + "\n")
                show_grid()
                print("Enter your move: {}\n".format(p1_moves[move_num1]))
                f.write("Enter your move: {}".format(p1_moves[move_num1]) +
"\n\n")
                current_move = p1_moves[move_num1].split(",")
                if len(current_move) < 2:
                    raise IndexError
                for item in current_move:
                    if item == "":
                        raise IndexError
                if current_move[1].isdigit():
                    raise ValueError
                assert int(p1_moves[move_num1][:-2]) < 11 and
p1_moves[move_num1][-1] in letters
                column1 = int(current_move[0]) - 1
                row1 = search_letters(current_move[1])
                position1 = p2_hidden_board[column1][row1]
                if position1 != "":
                    p2_board[column1][row1] = "X"
                    if position1 == "C":
                        ship_c2 +=1
                    elif position1 == "B":  # using OptionalPlayer2.txt
                        for i in optional_p2:
                            cur_move = i[1].split(",")
                            column = int(cur_move[0]) - 1
                            row = search_letters(cur_move[1])
                            if i[0][0] == "B":
                                if i[2] == "right":
                                    if p2_board[column][row] == "X" and
p2_board[column][row + 1] == "X" and \
                                            p2_board[column][row + 2] ==
"X" and p2_board[column][row + 3] == "X":
                                        ship_b2 += 1
                                        optional_p2.remove(i)

                                elif i[2] == "down":
                                    if p2_board[column][row] == "X" and
p2_board[column + 1][row] == "X" and \
                                            p2_board[column + 2][row] ==
"X" and p2_board[column + 3][row] == "X":
                                        ship_b2 += 1
                                        optional_p2.remove(i)

                    elif position1 == "D":
                        ship_d2 +=1
                    elif position1 == "S":
                        ship_s2 +=1
                    elif position1 == "P":
                        for i in optional_p2:
                            cur_move = i[1].split(",")
                            column = int(cur_move[0]) - 1
                            row = search_letters(cur_move[1])
                            if i[0][0] == "P":
                                if i[2] == "right":
                                    if p2_board[column][row] == "X" and
p2_board[column][row + 1] == "X":
                                        ship_p2 += 1
                                        optional_p2.remove(i)

                                elif i[2] == "down":
                                    if p2_board[column][row] == "X" and
p2_board[column + 1][row] == "X":
                                        ship_p2 += 1
                                        optional_p2.remove(i)

                else:
                    p2_board[column1][row1] = "O"
                control_ships_p1()
                move_num1 += 1
                if sum(i.count("X") for i in p2_board) == 27 and
sum(i.count("X") for i in p1_board) < 26:
                    print("Player1 Wins!\n")
                    f.write("Player1 Wins!"+"\n\n")
                    control_ships_p2()
                    control_ships_p1()
                    break
                player2()
            except IndexError:
                if len(current_move) == 1:
                    if current_move == [""]:
                        print("IndexError: You did not enter any move!\n")
                        f.write("IndexError: You did not enter any move!" +
"\n\n")
                    elif current_move[0].isdigit():
                        print("IndexError: You forgot to enter a comma and
a letter!\n")
                        f.write("IndexError: You forgot to enter a comma
and a letter!" + "\n\n")
                    else:
                        print("IndexError: You forgot to enter a number and
a comma!\n")
                        f.write("IndexError: You forgot to enter a number
and a comma!" + "\n\n")
                else:
                    if current_move[0] == "" and current_move[1] == "":
                        print("IndexError: You forgot to enter a number and
a letter!\n")
                        f.write("IndexError: You forgot to enter a number
and a letter!" + "\n\n")
```

```
                    elif current_move[0] != "" and current_move[1] == "":
                        print("IndexError: You forgot to enter a
letter!\n")
                        f.write("IndexError: You forgot to enter a letter!"
+ "\n\n")
                    elif current_move[0] == "" and current_move[1] != "":
                        print("IndexError: You forgot to enter a
number!\n")
                        f.write("IndexError: You forgot to enter a number!"
+ "\n\n")
                move_num1 += 1
                player1()
            except ValueError:
                if len(current_move) > 2:
                    print("ValueError: You entered too many characters!
Perhaps you forgot a semicolon?\n")
                    f.write("ValueError: You entered too many characters!
Perhaps you forgot a semicolon?" + "\n\n")
                elif not current_move[0].isdigit() and
current_move[1].isdigit():
                    print("ValueError: Please reverse the locations of the
letter and the number!\n")
                    f.write("ValueError: Please reverse the locations of
the letter and the number!" + "\n\n")
                elif current_move[0].isdigit() and
current_move[1].isdigit():
                    print("ValueError: You should enter a letter after
comma!\n")
                    f.write("ValueError: You should enter a letter after
comma!" + "\n\n")
                elif not current_move[0].isdigit() and not
current_move[1].isdigit():
                    print("ValueError: You should enter a number before
comma!\n")
                    f.write("ValueError: You should enter a number before
comma!" + "\n\n")
                move_num1 += 1
                player1()
            except AssertionError:
                print("AssertionError: Invalid Operation.\n")
                f.write("AssertionError: Invalid Operation.\n\n")
                move_num1 += 1
                player1()
            except:
                print("kaBOOM: run for your life!\n")
                f.write("kaBOOM: run for your life!"+"\n\n")
                move_num1 += 1
                player1()

    def player2():
        global round_num, move_num2, p1_ships, ship_c1, ship_b1, ship_d1,
ship_s1, ship_p1
        while move_num2 < min(len(p1_moves), len(p2_moves)):
            try:
                print("Player2's Move\n")
                print("Round : {}".format(round_num) + "\t\t" + "Grid Size:
10x10\n")
                print("Player1's Hidden Board" + "\t" + "Player2's Hidden
Board")
                print("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I
J")
```

```
                f.write("Player2's Move" + "\n\n")
                f.write("Round : {}".format(round_num) + "\t\t\t\t" + "Grid 
Size: 10x10" + "\n\n")
                f.write("Player1's Hidden Board" + "\t" + "Player2's Hidden 
Board" + "\n")
                f.write("  A B C D E F G H I J" + "\t" + "  A B C D E F G H 
I J" + "\n")
                show_grid()
                print("Enter your move: {}\n".format(p2_moves[move_num2]))
                f.write("Enter your move: {}".format(p2_moves[move_num2]) + 
"\n\n")
                current_move = p2_moves[move_num2].split(",")
                if len(current_move) < 2:
                    raise IndexError
                for item in current_move:
                    if item == "":
                        raise IndexError
                if current_move[1].isdigit():
                    raise ValueError
                assert int(p2_moves[move_num2][:-2]) < 11 and 
p2_moves[move_num2][-1] in letters
                column2 = int(current_move[0]) - 1
                row2 = search_letters(current_move[1])
                position2 = p1_hidden_board[column2][row2]

                if position2 != "":
                    p1_board[column2][row2] = "X"
                    if position2 == "C":
                        ship_c1 += 1
                    elif position2 == "B":
                        for i in optional_p1:
                            cur_move = i[1].split(",")
                            column = int(cur_move[0]) - 1
                            row = search_letters(cur_move[1])
                            if i[0][0] == "B":
                                if i[2] == "right":
                                    if p1_board[column][row] == "X" and 
p1_board[column][row + 1] == "X" and \
                                            p1_board[column][row + 2] == 
"X" and p1_board[column][row + 3] == "X":
                                        ship_b1 += 1
                                        optional_p1.remove(i)
                                elif i[2] == "down":
                                    if p1_board[column][row] == "X" and 
p1_board[column + 1][row] == "X" and \
                                            p1_board[column + 2][row] == 
"X" and p1_board[column + 3][row] == "X":
                                        ship_b1 += 1
                                        optional_p1.remove(i)
                    elif position2 == "D":
                        ship_d1 += 1
                    elif position2 == "S":
                        ship_s1 += 1
                    elif position2 == "P":
                        for i in optional_p1:
                            cur_move = i[1].split(",")
                            column = int(cur_move[0]) - 1
                            row = search_letters(cur_move[1])
                            if i[0][0] == "P":
                                if i[2] == "right":
                                    if p1_board[column][row] == "X" and 
```

```
p1_board[column][row + 1] == "X":
                                        ship_p1 += 1
                                        optional_p1.remove(i)
                                elif i[2] == "down":
                                    if p1_board[column][row] == "X" and
p1_board[column + 1][row] == "X":
                                        ship_p1 += 1
                                        optional_p1.remove(i)
                else:
                    p1_board[column2][row2] = "O"
                control_ships_p2()
                move_num2 += 1
                if sum(i.count("X") for i in p1_board) == 27 and
sum(i.count("X") for i in p2_board) < 27:
                    print("Player2 Wins!\n")
                    f.write("Player2 Wins!" + "\n\n")
                    control_ships_p1()
                    control_ships_p2()
                    break
                if sum(i.count("X") for i in p2_board) == 27 and
sum(i.count("X") for i in p1_board) == 27:
                    print("It is a Draw!\n")
                    f.write("It is a Draw!" + "\n\n")
                    control_ships_p2()
                    control_ships_p1()
                    break
                round_num += 1
                player1()
            except IndexError:
                if len(current_move) == 1:
                    if current_move == [""]:
                        print("IndexError: You did not enter any move!\n")
                        f.write("IndexError: You did not enter any move!" +
"\n\n")
                    elif current_move[0].isdigit():
                        print("IndexError: You forgot to enter a comma and
a letter!\n")
                        f.write("IndexError: You forgot to enter a comma
and a letter!" + "\n\n")
                    else:
                        print("IndexError: You forgot to enter a number and
a comma!\n")
                        f.write("IndexError: You forgot to enter a number
and a comma!" + "\n\n")
                else:
                    if current_move[0] == "" and current_move[1] == "":
                        print("IndexError: You forgot to enter a number and
a letter!\n")
                        f.write("IndexError: You forgot to enter a number
and a letter!" + "\n\n")
                    elif current_move[0] != "" and current_move[1] == "":
                        print("IndexError: You forgot to enter a
letter!\n")
                        f.write("IndexError: You forgot to enter a letter!"
+ "\n\n")
                    elif current_move[0] == "" and current_move[1] != "":
                        print("IndexError: You forgot to enter a
number!\n")
                        f.write("IndexError: You forgot to enter a number!"
+ "\n\n")
                move_num2 += 1
                player2()
            except ValueError:
                if len(current_move) > 2:
                    print("ValueError: You entered too many characters!
Perhaps you forgot a semicolon?\n")
                    f.write("ValueError: You entered too many characters!
Perhaps you forgot a semicolon?" + "\n\n")
                elif not current_move[0].isdigit() and
current_move[1].isdigit():
                    print("ValueError: Please reverse the locations of the
letter and the number!\n")
                    f.write("ValueError: Please reverse the locations of
the letter and the number!" + "\n\n")
                elif current_move[0].isdigit() and
current_move[1].isdigit():
                    print("ValueError: You should enter a letter after
comma!\n")
                    f.write("ValueError: You should enter a letter after
comma!" + "\n\n")
                elif not current_move[0].isdigit() and not
current_move[1].isdigit():
                    print("ValueError: You should enter a number before
comma!\n")
                    f.write("ValueError: You should enter a number before
comma!" + "\n\n")
                move_num2 += 1
                player2()
            except AssertionError:
                print("AssertionError: Invalid Operation.\n")
                f.write("AssertionError: Invalid Operation.\n\n")
                move_num2 += 1
                player2()
            except:
                print("kaBOOM: run for your life!\n")
                f.write("kaBOOM: run for your life!"+"\n\n")
                move_num2 += 1
                player2()
    player1()
    print("Final Information\n")
    print("Player1's Board" + "\t\t" + "  Player2's Board")
    print("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I J")
    f.write("Final Information"+"\n\n")
    f.write("Player1's Board" + "\t\t" + "    Player2's Board"+"\n")
    f.write("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I J"+"\n")
    for k in range(10):  # this for loop is for printing the hidden ship
boxes at the end of the game
        for p in range(10):
            if p1_board[k][p] == "-" and p1_hidden_board[k][p] != "":
                p1_board[k][p] = p1_hidden_board[k][p]
            if p2_board[k][p] == "-" and p2_hidden_board[k][p] != "":
                p2_board[k][p] = p2_hidden_board[k][p]
    show_grid()
```

Where to use this program ?

This game is a well-known, world-wide game. So, anybody who wants to play Battleship game do not need a paper and a pen to play.

Showing the grid function basically can be used for any table that needs to be printed.

Description of the Program

Importing Modules and Arranging Lists/Variables

```
import os
import sys

error_file = []  # if any error occurs while trying to open files, they
will be added to this list
p1_hidden_board = []
p2_hidden_board = []
p1_moves = []
p2_moves = []
p1_board = []
p2_board = []
p1_ships = ["-", "- -", "-", "-", "- - - -"]
p2_ships = ["-", "- -", "-", "-", "- - - -"]
# sinked ship number of player1
ship_c1, ship_b1, ship_d1, ship_s1, ship_p1 = 0, 0, 0, 0, 0
# sinked ship number of player2
ship_c2, ship_b2, ship_d2, ship_s2, ship_p2 = 0, 0, 0, 0, 0
```

I arranged lists and variables that will be used later in this program. “p1” named variables are used for player1 and “p2” named variables used for player2.

Reading Files

```
current_dir_path = os.getcwd()

reading_file_name = "OptionalPlayer1.txt"
reading_file_path = os.path.join(current_dir_path, reading_file_name)
with open(reading_file_path, "r") as f:
    optional_p1 = f.read().split("\n")
    for i in range(len(optional_p1)):
        optional_p1[i] = optional_p1[i].replace(":",";")
        optional_p1[i] = optional_p1[i].split(";")
reading_file_name = "OptionalPlayer2.txt"
reading_file_path = os.path.join(current_dir_path, reading_file_name)
with open(reading_file_path, "r") as f:
    optional_p2 = f.read().split("\n")
    for i in range(len(optional_p2)):
        optional_p2[i] = optional_p2[i].replace(":",";")
        optional_p2[i] = optional_p2[i].split(";")
try:
    reading_file_name = sys.argv[1]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.readline().split("\n")
        while line_data != [""]:  # the program reads input file line by
line until the empty line
            if len(line_data) == 2:
                line_data.remove(line_data[1])
            line_data[0] = line_data[0].split(";")
            p1_hidden_board.extend(line_data)
            line_data = f.readline().split("\n")
except IOError:
    error_file.append(reading_file_name)

try:
    reading_file_name = sys.argv[2]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.readline().split("\n")
        while line_data != [""]:  # the program reads input file line by
line until the empty line
            if len(line_data) == 2:
                line_data.remove(line_data[1])
            line_data[0] = line_data[0].split(";")
            p2_hidden_board.extend(line_data)
            line_data = f.readline().split("\n")
except IOError:
    error_file.append(reading_file_name)

try:
    reading_file_name = sys.argv[3]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.read()
        line_data = line_data.split(";")
        p1_moves.extend(line_data)
except IOError:
    error_file.append(reading_file_name)

try:
    reading_file_name = sys.argv[4]
    reading_file_path = os.path.join(current_dir_path, reading_file_name)
    with open(reading_file_path, "r") as f:
        line_data = f.read()
        line_data = line_data.split(";")
        p2_moves.extend(line_data)

        # with for loop down below, we create the game board for both
players
        for i in range(10):
            row = []
            for j in range(10):
                row.append("-")
            p1_board.append(row)
        for i in range(10):  # I wrote separate for loops because
            row = []  # if I write them in the same loop they would
represent the same list
            for j in range(10):
                row.append("-")
            p2_board.append(row)
        p1_moves = p1_moves[:-1]
        p2_moves = p2_moves[:-1]

        letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]

        def search_letters(move):  # function that returns the index of
letters
            for letter in letters:
                if letter == move:
                    return letters.index(letter)
except IOError:
    error_file.append(reading_file_name)
```

For the first five files, the program reads the file and adds necessary informations to lists. It does the same thing for the sixth file but also it creates the game boards and defines the function that returns the index of the letters. Additionally, every file is checked whether it can or cannot be opened. If there is/are file(s) that cannot be opened, the program records the file's name inside a list to later print it.

Writing to File

```
writing_file_name = "Battleship.out"
writing_file_path = os.path.join(current_dir_path, writing_file_name)
with open(writing_file_path, "w") as f:
    if len(error_file) != 0:  # if at least one file cannot be opened,
program writes IOError
        names = ""
        for name in error_file:
            names = names + name + ","
        names = names[:-1]  # removing last comma at the end
        print("IOError: input file(s) {} is/are not
reachable.".format(names))
        f.write("IOError: input file(s) {} is/are not
reachable.\n".format(names))
        exit()
    print("Battle of Ships Game\n")
    f.write("Battle of Ships Game\n\n")
    move_num1 = 0
    move_num2 = 0
    round_num = 1
```

We open a file to write into it. Then, we check if there is an error occured while opening a file. If there is, the program prints the files that cannot be opened and stops. If not, it prints the title and defines necessary variables.

Showing the Grids Function

```
def show_grid():
    for i in range(10):
        print(i + 1, end=" ")
        f.write(str(i+1)+" ")
        for j in range(10):
            print(p1_board[i][j], end=" ")
            f.write(p1_board[i][j]+" ")
        print("\t" + str(i + 1), end=" ")
        f.write("\t"+str(i+1)+" ")
        for j in range(10):
            print(p2_board[i][j], end=" ")
            f.write(p2_board[i][j]+" ")
        print()
        f.write("\n")
    print()
    print("Carrier\t\t"+p1_ships[0]+"\tCarrier\t\t"+p2_ships[0])
    print("Battleship\t" + p1_ships[1] + "\tBattleship\t" + p2_ships[1])
    print("Destroyer\t" + p1_ships[2] + "\tDestroyer\t" + p2_ships[2])
    print("Submarine\t" + p1_ships[3] + "\tSubmarine\t" + p2_ships[3])
    print("Patrol Boat\t" + p1_ships[4] + "\tPatrol Boat\t" + p2_ships[4])
    print()
    f.write("\n")
    f.write("Carrier\t\t" + p1_ships[0] + "\t\t\tCarrier\t\t" +
p2_ships[0]+"\n")
    f.write("Battleship\t" + p1_ships[1] + "\t\t\tBattleship\t" +
p2_ships[1]+"\n")
    f.write("Destroyer\t" + p1_ships[2] + "\t\t\tDestroyer\t" +
p2_ships[2]+"\n")
    f.write("Submarine\t" + p1_ships[3] + "\t\t\tSubmarine\t" +
p2_ships[3]+"\n")
    f.write("Patrol Boat\t" + p1_ships[4] + "\t\tPatrol Boat\t" +
p2_ships[4]+"\n")
    f.write("\n")
```

Show grid function prints two boards and the informations about ships.

Controlling the Ships Functions

```
def control_ships_p1():  # shows sinked ships based on sinked boxes
    if ship_c2 == 5:
        p2_ships[0] = "X"
    if ship_b2 == 1:
        p2_ships[1] = "X -"
    elif ship_b2 == 2:
        p2_ships[1] = "X X"
    if ship_d2 == 3:
        p2_ships[2] = "X"
    if ship_s2 == 3:
        p2_ships[3] = "X"
    if ship_p2 == 1:
        p2_ships[4] = "X - - -"
    elif ship_p2 == 2:
        p2_ships[4] = "X X - -"
    elif ship_p2 == 3:
        p2_ships[4] = "X X X -"
    elif ship_p2 == 4:
        p2_ships[4] = "X X X X"
```

```
def control_ships_p2():
    if ship_c1 == 5:
        p1_ships[0] = "X"
    if ship_b1 == 1:
        p1_ships[1] = "X -"
    elif ship_b1 == 2:
        p1_ships[1] = "X X"
    if ship_d1 == 3:
        p1_ships[2] = "X"
    if ship_s1 == 3:
        p1_ships[3] = "X"
    if ship_p1 == 1:
        p1_ships[4] = "X - - -"
    elif ship_p1 == 2:
        p1_ships[4] = "X X - -"
    elif ship_p1 == 3:
        p1_ships[4] = "X X X -"
    elif ship_p1 == 4:
        p1_ships[4] = "X X X X"
```

These two functions serve same purpose. They check the number of sinked ships and according to that they change the ship informations.

Player’s Functions

Since two players’ functions work the same, I will only explain one of them.

```
def player1():
    global round_num, move_num1, p2_ships, ship_c2, ship_b2, ship_d2,
ship_s2, ship_p2
    while move_num1 < min(len(p1_moves), len(p2_moves)):
        try:
            print("Player1's Move\n")
            print("Round : {}".format(round_num) + "\t\t" + "Grid Size:
10x10\n")
            print("Player1's Hidden Board" + "\t" + "Player2's Hidden
Board")
            print("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I J")
            f.write("Player1's Move" + "\n\n")
            f.write("Round : {}".format(round_num) + "\t\t\t\t" + "Grid
Size: 10x10" + "\n\n")
            f.write("Player1's Hidden Board" + "\t" + "Player2's Hidden
Board" + "\n")
            f.write("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I
J" + "\n")
            show_grid()
            print("Enter your move: {}\n".format(p1_moves[move_num1]))
            f.write("Enter your move: {}".format(p1_moves[move_num1]) +
"\n\n")
```

Firstly, we specify the lists and variables as global. We want the program to run until the last move of the player, so we use while loop. Inside try block, we first print and write to file the information about moves, rounds and hidden boards.

```
current_move = p1_moves[move_num1].split(",")
if len(current_move) < 2:
    raise IndexError
for item in current_move:
    if item == "":
        raise IndexError
if current_move[1].isdigit():
    raise ValueError
assert int(p1_moves[move_num1][:-2]) < 11 and p1_moves[move_num1][-1] in
letters
```

We read the moves one by one and check if there is an error. We will see the results of exceptions a bit later.

```
column1 = int(current_move[0]) - 1
row1 = search_letters(current_move[1])
position1 = p2_hidden_board[column1][row1]
if position1 != "":
    p2_board[column1][row1] = "X"
    if position1 == "C":
        ship_c2 +=1
    elif position1 == "B":  # using OptionalPlayer2.txt
        for i in optional_p2:
            cur_move = i[1].split(",")
            column = int(cur_move[0]) - 1
            row = search_letters(cur_move[1])
            if i[0][0] == "B":
                if i[2] == "right":
                    if p2_board[column][row] == "X" and
p2_board[column][row + 1] == "X" and \
                            p2_board[column][row + 2] == "X" and
p2_board[column][row + 3] == "X":
                        ship_b2 += 1
                        optional_p2.remove(i)

                elif i[2] == "down":
                    if p2_board[column][row] == "X" and p2_board[column +
1][row] == "X" and \
                            p2_board[column + 2][row] == "X" and
p2_board[column + 3][row] == "X":
                        ship_b2 += 1
                        optional_p2.remove(i)

    elif position1 == "D":
        ship_d2 +=1
    elif position1 == "S":
        ship_s2 +=1
    elif position1 == "P":
        for i in optional_p2:
            cur_move = i[1].split(",")
            column = int(cur_move[0]) - 1
            row = search_letters(cur_move[1])
            if i[0][0] == "P":
                if i[2] == "right":
                    if p2_board[column][row] == "X" and
p2_board[column][row + 1] == "X":
                        ship_p2 += 1
                        optional_p2.remove(i)

                elif i[2] == "down":
                    if p2_board[column][row] == "X" and p2_board[column +
1][row] == "X":
                        ship_p2 += 1
                        optional_p2.remove(i)

else:
    p2_board[column1][row1] = "O"
control_ships_p1()
move_num1 += 1
```

We arrange the column and row according to current move and name it position1. After that, we check if there is ship on the given position. If there is a ship piece, we check the class of ship and remaining ships. For ships C, D, S we only check the number of sinked pieces since there is one of them. But for B and P we have to check the positions. At the end, we renew the information about sinked ships.

```
if sum(i.count("X") for i in p2_board) == 27 and sum(i.count("X") for i in
p1_board) < 26:
    print("Player1 Wins!\n")
    f.write("Player1 Wins!"+"\n\n")
    control_ships_p2()
    control_ships_p1()
    break
player2()
```

Wit this if loop, we check if player1 wins. At the end of the try block, we call the player2's function because if there is no error, this means that we can move on to other player.

```
except IndexError:
    if len(current_move) == 1:
        if current_move == [""]:
            print("IndexError: You did not enter any move!\n")
            f.write("IndexError: You did not enter any move!" + "\n\n")
        elif current_move[0].isdigit():
            print("IndexError: You forgot to enter a comma and a
letter!\n")
            f.write("IndexError: You forgot to enter a comma and a letter!"
+ "\n\n")
        else:
            print("IndexError: You forgot to enter a number and a
comma!\n")
            f.write("IndexError: You forgot to enter a number and a comma!"
+ "\n\n")
    else:
        if current_move[0] == "" and current_move[1] == "":
            print("IndexError: You forgot to enter a number and a
letter!\n")
            f.write("IndexError: You forgot to enter a number and a
letter!" + "\n\n")
        elif current_move[0] != "" and current_move[1] == "":
            print("IndexError: You forgot to enter a letter!\n")
            f.write("IndexError: You forgot to enter a letter!" + "\n\n")
        elif current_move[0] == "" and current_move[1] != "":
            print("IndexError: You forgot to enter a number!\n")
            f.write("IndexError: You forgot to enter a number!" + "\n\n")
    move_num1 += 1
    player1()
```

```
except ValueError:
    if len(current_move) > 2:
        print("ValueError: You entered too many characters! Perhaps you 
forgot a semicolon?\n")
        f.write("ValueError: You entered too many characters! Perhaps you 
forgot a semicolon?" + "\n\n")
    elif not current_move[0].isdigit() and current_move[1].isdigit():
        print("ValueError: Please reverse the locations of the letter and 
the number!\n")
        f.write("ValueError: Please reverse the locations of the letter and 
the number!" + "\n\n")
    elif current_move[0].isdigit() and current_move[1].isdigit():
        print("ValueError: You should enter a letter after comma!\n")
        f.write("ValueError: You should enter a letter after comma!" + 
"\n\n")
    elif not current_move[0].isdigit() and not current_move[1].isdigit():
        print("ValueError: You should enter a number before comma!\n")
        f.write("ValueError: You should enter a number before comma!" + 
"\n\n")
    move_num1 += 1
    player1()
except AssertionError:
    print("AssertionError: Invalid Operation.\n")
    f.write("AssertionError: Invalid Operation.\n\n")
    move_num1 += 1
    player1()
except:
    print("kaBOOM: run for your life!\n")
    f.write("kaBOOM: run for your life!"+"\n\n")
    move_num1 += 1
    player1()
```

We can see all the prints of the errors. Since the program needs to continue to read same player's input even after an error occurs, we increase the move number and call the same player's function at the end.

```
if sum(i.count("X") for i in p1_board) == 27 and sum(i.count("X") for i in 
p2_board) < 27:
    print("Player2 Wins!\n")
    f.write("Player2 Wins!" + "\n\n")
    control_ships_p1()
    control_ships_p2()
    break
if sum(i.count("X") for i in p2_board) == 27 and sum(i.count("X") for i in 
p1_board) == 27:
    print("It is a Draw!\n")
    f.write("It is a Draw!" + "\n\n")
    control_ships_p2()
    control_ships_p1()
    break
```

Player2's function is the same except the part you see above. We check if the player2 wins or that it is a draw in player2 function.

```
player1()
print("Final Information\n")
print("Player1's Board" + "\t\t" + "  Player2's Board")
print("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I J")
f.write("Final Information"+"\n\n")
f.write("Player1's Board" + "\t\t" + "    Player2's Board"+"\n")
f.write("  A B C D E F G H I J" + "\t" + "  A B C D E F G H I J"+"\n")
for k in range(10):  # this for loop is for printing the hidden ship boxes
at the end of the game
    for p in range(10):
        if p1_board[k][p] == "-" and p1_hidden_board[k][p] != "":
            p1_board[k][p] = p1_hidden_board[k][p]
        if p2_board[k][p] == "-" and p2_hidden_board[k][p] != "":
            p2_board[k][p] = p2_hidden_board[k][p]
show_grid()
```

After defining the functions, we call them above. Lastly, we print the final board with hidden ship pieces.

User's Catalogue

Program's user manual/tutorial

1.Each player will enter the positions of their ships secretly into Player1.txt and Player2.txt.

2.Each player will write their moves into Player1.in and Player2.in.

3.Each player will write the positions of class B and P ships into OptionalPlayer1.txt and OptionalPlayer2.txt.

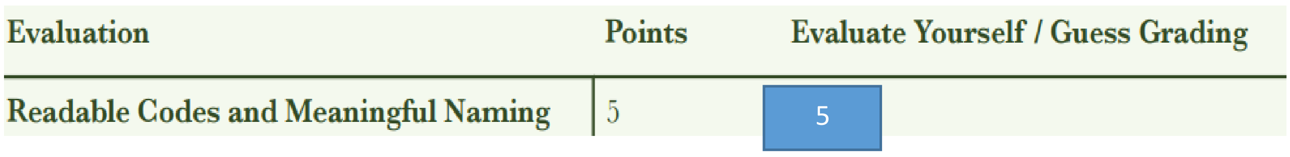
4.Run the Assignment4.py program.

5.Go to Battleship.out and see the results.

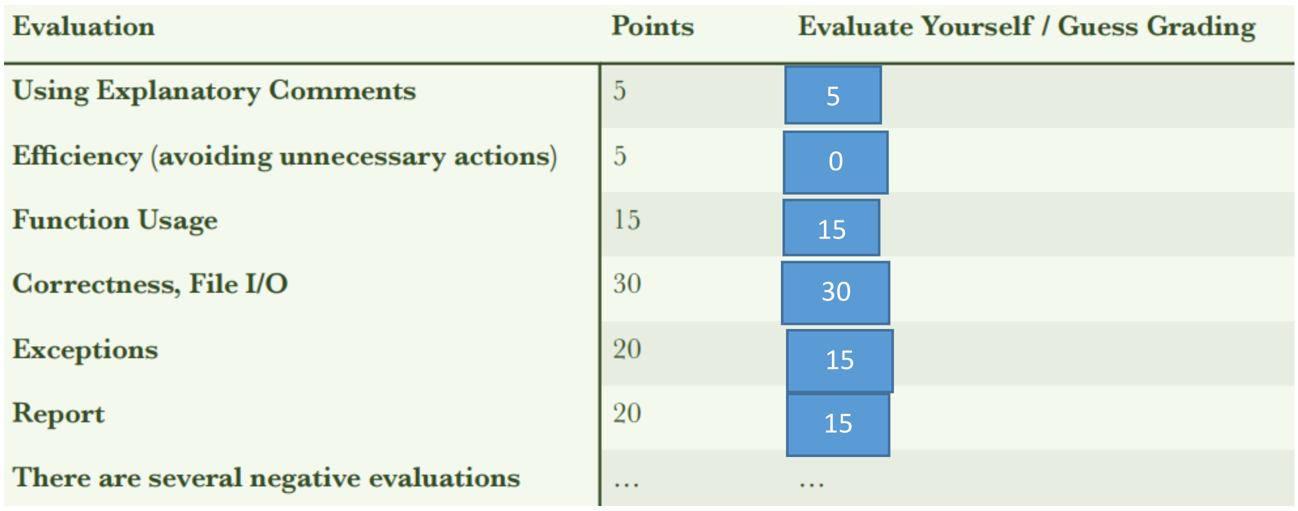
Restrictions on the program

Possible errors for OptionalPlayer1.txt and OptionalPlayer2.txt are ignored.

Possible errors other than IOError for the files taken from the command line are ignored.



Evaluation	Points	Evaluate Yourself / Guess Grading
Readable Codes and Meaningful Naming	5	5



Evaluation	Points	Evaluate Yourself / Guess Grading
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	0
Function Usage	15	15
Correctness, File I/O	30	30
Exceptions	20	15
Report	20	15
There are several negative evaluations	…	…