Hacettepe University
Departmant Of Computer Engineeering

BBM 103 Assignment 4 Report

Emirhan Utku-2210765029

02.01.2023

CONTENTS:

1)ANALYSİS

In this problem, we were asked to make an admiral sank game for two players whose moves were given to us.

The admiral sunk is a classic two-player game of thought and strategy. Each player hides their ship's floor from the opposing player, and the game is played on an area with predetermined horizontal and vertical lines.

The game starts from player 1 and asks him to take a shot. If player 1 hits any part of the opponent's ships, the letter "X" appears in the table, if he cannot hit, the letter "O" appears in the table. The same events are repeated for the 2nd player. The game continues until all of a player's ship pieces are sunk.

Information given after each move:

- Hidden table of players 1 and 2
- What round are we in
- Which player will move and that player's move
- Current status of sunk and unsinkable ships

2)DESİGN

First the ships of player 1 are taken as the first argument in the terminal, the ships of the second player as the second argument, the moves of the 1st player as the 3rd argument and the moves of the 2nd player as the 4th argument. The accuracy of these documents is checked with try except.

The "required_inputs()" function allows us to put the information we receive from the players into the lists we call all_ships1,all_ships2,moves1 and moves2.

The "grids()" function allows us to create the tables of players 1 and 2 using the lists all_ships1 and all_ships2 that we created above, and create the hidden tables that we will use while playing the game.

The "ships()" function allows us to insert the coordinates of other ships, except the petrol boat and the battle ship, into the dictionaries we call ships1 and ships2.

The "optional_txt_func()" function allows us to insert the coordinates of the petrol boat and battleship ships into ships1 and ships2 by using the optional txts given from outside.

The "write_table()" function checks the status of the ships and shows them whether they are sunk or not. If all parts of the ship have not sunk, "-" is written next to it, if it has sunk, it is written "X".

The "write_tables()" function allows us to print hidden boards using main_table dictionaries.

The "control_of_moves()" function allows us to find and delete the part of a ship from the ships1 and ships2 dictionary if a part of a ship has been hit.

The "error_func()" function checks for possible errors.

The "moves()" function prints the text that must be written at the beginning of each move, and the "game()" function plays the game

3)Programmer's Catalogue

3.1) Required_inputs()

```
def required_inputs():
    global moves1, moves2, all_ships1, all_ships2, ships1, ships2
    all_ships1 = []
    all_ships2 = []
    moves2 = player2_moves_input.readline().rstrip(";").split(";")
    moves1 = player1_moves_input.readline().rstrip(";").split(";")
    for i in range(len(open(player1_ships, "r").readlines())):
        ships1 = player1_ships_input.readline().rstrip("\n").split(";")
        ships2 = player2_ships_input.readline().rstrip("\n").split(";")
        all_ships1.append(ships1)
        all_ships2.append(ships2)
```

We separated the contents of the player.in files from the semicolons and put them in the move's lists

We took what was written in the player.txt files line by line and separated them from semicolons inside the for loop and put them into the ships lists

3.2) Grids()

```
def grids():
    for i in range(10):
        Player1s_Board[i + 1] = {}
        Player2s_Board[i + 1] = {}
        Main_Table1[i + 1] = {}
        Main_Table2[i + 1] = {}
        for k in range(len(letters)):
            Player1s_Board[i + 1][letters[k]] = {}
            Player2s_Board[i + 1][letters[k]] = {}
            Main_Table1[i + 1][letters[k]] = {}
            Main_Table2[i + 1][letters[k]] = {}
            Main_Table1[i + 1][letters[k]] = "-"
            Main_Table2[i + 1][letters[k]] = "-"
            Player1s_Board[i + 1][letters[k]] = all_ships1[i][k]
            Player2s_Board[i + 1][letters[k]] = all_ships2[i][k]
```

In the first of 2 for loops, we created row dictionaries inside the players_board dictionaries. In the second for loop, we set the keys of this row dictionary as columns and its values as the initial letter of the ship in that cell.

We created the main_table dictionaries in the same way, but we set all the values in this dictionary as "-" sign.

3.3) Ships()

```
def ships():
    global ships1, ships2
    for n in ("Carrier", "Destroyer", "Submarine"):
        ships1[n] = {}
        ships2[n] = {}
    for i in range(10):
        for l in ("Carrier", "Destroyer", "Submarine"):
            ships1[l][i + 1] = {}
            ships2[l][i + 1] = {}
        for k in range(len(letters)):
            if Player1s_Board[i + 1][letters[k]] == "C":
                ships1["Carrier"][i + 1][letters[k]] = "C"
            elif Player1s_Board[i + 1][letters[k]] == "D":
                ships1["Destroyer"][i + 1][letters[k]] = "D"
            elif Player1s_Board[i + 1][letters[k]] == "S":
                ships1["Submarine"][i + 1][letters[k]] = "S"
            if Player2s_Board[i + 1][letters[k]] == "C":
                ships2["Carrier"][i + 1][letters[k]] = "C"
            elif Player2s_Board[i + 1][letters[k]] == "D":
                ships2["Destroyer"][i + 1][letters[k]] = "D"
            elif Player2s_Board[i + 1][letters[k]] == "S":
                ships2["Submarine"][i + 1][letters[k]] = "S"
```

We opened another dictionary containing ship names in ships dictionaries, reopened column dictionary in this dictionary, set the keys of this column dictionary as row and made the values with the initial letter of that ship.

3.4) Optional_txt_func():

```
def optional_txt_func(optTxt, ships):
    opt_file = open(optTxt, "r")
    while True:
        line = opt_file.readline()
        if line == "":
            break
        else:

            ship_name, data = line.split(":")
            position, direction = data.split(";")[:-1]
            row, column = position.split(",")
            row = int(row)
            ship_lenghth = {"B": 4, "P": 2}
            ships[ship_name] = {}
            if direction == "right":
                column_index = letters.index(column)
                interval = letters[column_index:ship_lenghth[ship_name[0]] + column_index]
                ships[ship_name][row] = {}
                for column_counter in interval:
                    ships[ship_name][row][column_counter] = ship_name[0]
            elif direction == "down":
                interval = range(row, ship_lenghth[ship_name[0]] + row)
                for row_counter in interval:
                    ships[ship_name][row_counter] = {column: ship_name[0]}
    return ships
```

If "right" is written in the coordinate given in the optional files, we move the column keys into the same row dictionary as the length of the ship and make the values the initial letter of that ship.

If it says "down" in the coordinate given in the optional files, we increase the number of row dictionaries in the ship dictionary and leave the keys and values in it the same.

3.5) Write_table()

```
def write_table():
    for o in ("Carrier", "Battleship", "Destroyer", "Submarine", "Patrol Boat"):
        if o=="Carrier":
            print(o, end="\t\t")
            all_outputs.write(o+"\t\t")
        else:
            print(o, end="\t")
            all_outputs.write(o + "\t")
        if o == "Battleship":
            a = 0
            b = 0
            if any(list(ships1["B1"][l].values())[m] != "" for l in list(ships1["B1"].keys()) for m
                   range(len(list(ships1["B1"][l].values())))):
                a = a + 1
            else:
                b = b + 1
            if any(list(ships1["B2"][l].values())[m] != "" for l in list(ships1["B2"].keys()) for m
                   range(len(list(ships1["B2"][l].values())))):
                a = a + 1
            else:
                b = b + 1
            if a==0:
                print((b * "X ").strip(" "), end="")
                all_outputs.write((b * "X ").rstrip(" ") + "")
            else:
                print(b * "X ", end="")
                all_outputs.write(b * "X " + "")
            print((a * "- ").strip(""), end="\t\t\t\t")
            all_outputs.write((a * "- ").rstrip(" ") + "\t\t\t\t")
        elif o == "Patrol Boat":
            c = 0
            d = 0
            if any(list(ships1["P1"][l].values())[m] != "" for l in list(ships1["P1"].keys()) for m
                   range(len(list(ships1["P1"][l].values())))):
                c = c + 1
```

```
                d = d + 1
            if any(list(ships1["P2"][l].values())[m] != "" for l in list(ships1["P2"].keys()) for m
                   range(len(list(ships1["P2"][l].values())))):
                c = c + 1
            else:
                d = d + 1
            if any(list(ships1["P3"][l].values())[m] != "" for l in list(ships1["P3"].keys()) for m
                   range(len(list(ships1["P3"][l].values())))):
                c = c + 1
            else:
                d = d + 1
            if any(list(ships1["P4"][l].values())[m] != "" for l in list(ships1["P4"].keys()) for m
                   range(len(list(ships1["P4"][l].values())))):
                c = c + 1
            else:
                d = d + 1
            if c==0:
                print((d * "X ").rstrip(" "), end="")
                all_outputs.write((d * "X ").rstrip(" ")+"")
            else:
                print(d * "X ", end="")
                all_outputs.write(d * "X " + "")
            print((c * "- ").rstrip(" "), end="\t\t\t")
            all_outputs.write((c * "- ").rstrip(" ")+"\t\t\t")

        else:
            if any(list(ships1[o][l].values())[m] != "" for l in list(ships1[o].keys()) for m in
                   range(len(list(ships1[o][l].values())))):
                print("-", end="\t\t\t\t")
                all_outputs.write("-"+"\t\t\t\t")
            else:
                print("X", end="\t\t\t\t")
                all_outputs.write("X"+"\t\t\t\t")
        if o=="Carrier":
            print(o, end="\t\t")
```

Since the space between the signs, we will put on the sides of the ships changes according to the ship names, we first checked it and then we decided how many "-" and how many "X" signs we would put by looking inside the ships dictionaries. We did the same for Player 2 in the rest of the code.

3.6) **Write_tables()**

```
def write_tables():
    for i in range(10):
        if list(Main_Table1.keys())[i] > 9:

            print(list(Main_Table1.keys())[i], end="")
            all_outputs.write(str(list(Main_Table1.keys())[i])+"")
            for k in range(len(letters)):
                if k == 9:
                    print(list(Main_Table1[i + 1].values())[k], end="")
                    all_outputs.write(list(Main_Table1[i + 1].values())[k] + "")
                else:
                    print(list(Main_Table1[i + 1].values())[k], end=" ")
                    all_outputs.write(list(Main_Table1[i + 1].values())[k] + " ")

            print("\t\t"+list(Main_Table2.keys())[i], end="")
            all_outputs.write("\t\t"+str(list(Main_Table2.keys())[i])+"")
            for k in range(len(letters)):
                if k==9:
                    print(list(Main_Table2[i + 1].values())[k], end="")
                    all_outputs.write(list(Main_Table2[i + 1].values())[k]+"")
                else:
                    print(list(Main_Table2[i + 1].values())[k], end=" ")
                    all_outputs.write(list(Main_Table2[i + 1].values())[k] + " ")
            saving_outputs_to_file("")
            saving_outputs_to_file("")
            write_table()



        else:
            print(list(Main_Table1.keys())[i], end=" ")
            all_outputs.write(str(list(Main_Table1.keys())[i])+" ")
            for k in range(len(letters)):
                if k==9:
                    print(list(Main_Table1[i + 1].values())[k], end="")
                    all_outputs.write(list(Main_Table1[i + 1].values())[k]+"")
```

Since it is different for the 10th row whether there is a space next to the row number or not, we added an if block first. Using the values of our Main_tables dictionary, we determined the signs in the column corresponding to the row in the table.

3.7) Control_of_moves()

```
def control_of_moves(Table, moves, ships, board, j):
    Table[int(moves[j].split(",")[0])][moves[j].split(",")[1]] = "X"
    if board[int(moves[j].split(",")[0])][moves[j].split(",")[1]] == "B":
        if int(moves[j].split(",")[0]) in list(ships["B1"].keys()) and (moves[j].split(",")[1]) in li
                ships["B1"][int(moves[j].split(",")[0])].keys()):
            ships["B1"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
        else:
            ships["B2"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
    elif board[int(moves[j].split(",")[0])][moves[j].split(",")[1]] == "P":
        if int(moves[j].split(",")[0]) in list(ships["P1"].keys()) and (moves[j].split(",")[1]) in li
                ships["P1"][int(moves[j].split(",")[0])].keys()):
            ships["P1"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
        elif int(moves[j].split(",")[0]) in list(ships["P2"].keys()) and (moves[j].split(","))[1] in
                ships["P2"][int(moves[j].split(",")[0])].keys()):
            ships["P2"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
        elif int(moves[j].split(",")[0]) in list(ships["P3"].keys()) and (moves[j].split(","))[1] in
                ships["P3"][int(moves[j].split(",")[0])].keys()):
            ships["P3"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
        else:
            ships["P4"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
    elif board[int(moves[j].split(",")[0])][moves[j].split(",")[1]] == "C":
        ships["Carrier"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
    elif board[int(moves[j].split(",")[0])][moves[j].split(",")[1]] == "D":
        ships["Destroyer"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
    else:
        ships["Submarine"][int(moves[j].split(",")[0])][moves[j].split(",")[1]] = ""
```

Only when a part of any ship is hit, this function works and the value of the part corresponding to the coordinate in the dictionary of the ship's name in the ships dictionary becomes blank.

3.8) Error_func()

```
def value_error(positon):
    try:
        row=positon.split(",")[0]
        column=positon.split(",")[1]
        if len(positon.split(","))>2:
            saving_outputs_to_file("ValueError:Please enter a correct move\n")
            error_of_value=True
        elif row in alphabet and column not in alphabet:
            saving_outputs_to_file("ValueError:Row must be number and column must be letter\n")
            error_of_value=True
        elif column not in alphabet:
            saving_outputs_to_file("ValueError:Column must be letter\n")
            error_of_value = True
        elif row in alphabet:
            saving_outputs_to_file("ValueError:Row must be number\n")
            error_of_value = True
        else:
            error_of_value=False
        return error_of_value
    except:
        saving_outputs_to_file("kaBOOM: run for your life!\n")
        error_of_value=True
        return error_of_value
```

There is more than one function inside this function. The picture I used above is just one of these functions.

Before writing the code, we had the possible problems given to us checked inside the if/elif/else blocks, if there is a problem, we changed the error_of_.... value at the end of these blocks to true, if there is no problem, the error_of_.... value will remain false.

3.9) Moves()

```
def moves(a, j):
    saving_outputs_to_file(a+ "\n")
    saving_outputs_to_file("Round : {}\t\t\t\t\tGrid Size: 10x10\n".format(j + 1))
    saving_outputs_to_file("Player1's Hidden Board\t\tPlayer2's Hidden Board")
    saving_outputs_to_file("  "+ " ".join(letters)+ "\t\t"+ "  "+"  "+ " ".join(letters))
    write_tables()
```

We collect the number of rounds, grid size, table names and column names of the players in a single function and run this function before each move. By using the ".join()" function, we can print the elements in a list by leaving a space between them.

3.10) Game()

```
    while True:
        if currentCounter == len(current_player_moves):
            result1(currentCounter)
            result2()

            quit()
        if error_func(current_player_moves[currentCounter]) != True:
            if currentMainTable[int(current_player_moves[currentCounter].split(",")[0])][current_player_moves[currentCounter].spli
                saving_outputs_to_file("AssertionError: Invalid Operation\n")
                current_player_moves.pop(currentCounter)
                game_for_player(current_player_moves, next_player_moves, currentCounter, nextCounter)
            if currentMainTable[int(current_player_moves[currentCounter].split(",")[0])][current_player_moves[currentCounter].spli
                saving_outputs_to_file("AssertionError: Invalid Operation\n")
                current_player_moves.pop(currentCounter)
                game_for_player(current_player_moves, next_player_moves, currentCounter, nextCounter)
            moves(**moves_arguments)
            if currentBoard[int(current_player_moves[currentCounter].split(",")[0])][current_player_moves[currentCounter].split(",
                saving_outputs_to_file("Enter your move: {}\n".format(current_player_moves[currentCounter]))
                currentMainTable[int(current_player_moves[currentCounter].split(",")[0])][current_player_moves[currentCounter].spl
            else:
                saving_outputs_to_file("Enter your move: {}\n".format(current_player_moves[currentCounter]))
                control_of_moves(**com_arguments)

            currentCounter += 1


            game_for_player(next_player_moves, current_player_moves, nextCounter, currentCounter)
        else:
            current_player_moves.pop(currentCounter)
            game_for_player(current_player_moves, next_player_moves, currentCounter, nextCounter)

game_for_player()
```

First, we check the error_of_.... value that we use inside the "error_func()" function. If this value is "True", that is, if there is any problem in the entered moves, we enter the relevant function in error_func() and print what it should write. Then we delete the problematic move and play the game again for the same player.

If this value is "False", it first checks whether the next move has been played before, if it has been played, it throws AssertionError and plays the current player again. If there is no problem, we check whether the played move hit a ship, if it was hit, we print "X" and run the control_of_moves() function, if not, we print "O" and continue the game.

Time spent analyzing, designing, implementing, testing, and reporting

This homework was one of the homeworks that I enjoyed very much. It took about two and a half weeks because of its detail, and I worked on it for almost 4 hours a day. At the end of this assignment, I better understood how to deal with possible user-related problems, namely the use of try/except.

Reusability

Programmers can use the code I wrote as they wish. Because I think my code is very clear and not very complex

4)User Catalogue

To use this program, both players must first have their moves in a file with the .in extension. After that, the locations of the ships of both players should be in the .txt file, and the coordinates of the ships "Battleship" and "PetrolBoat" should be in the OptinalPlayer1, 2.txt files. We should keep all these files in the same file as the python code we wrote. After running the code, our output will be in the Battleship file with the .out extension. By opening this file, you can check who won the game and whether you hit the opponent's ship by going to the number of rounds of the move you played.

Restrictions on the program

- To play the game, you must enter the coordinates of the ships and your moves in the specified order.
- Due to the rule of the game the number of correct moves entered must be equal for both players

Grading Table

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	5
Function Usage	15	15
Correctness, File I/O	30	30
Exceptions	20	20
Report	20	20
There are several negative evaluations	…	0