COURSE NAME:INTRODUCTION TO PROGRAMMING LAB.

ASSIGNMENT NUMBER: 4

STUDENT NUMBER : 2210356066

NAME/SURNAME:ZEYNEP NİSA KARATAŞ

DELİVERY DATE:30/12/2022

CONTENTS

ANALYSIS

Battleship is a well-known game among the world. At first I would like to tell how the game works and the strategy of the game. Battleship is a guessing game for two players. Each player makes his/her own battleship board.Battleship boards are 10x10 board so there are 100 squares. Each player has to have 5 different type of boards and they have to specify the ships places and the ships shouldn't overlap. For the types of ships there are Carrier, Battleship, Destroyer, Submarine and Patrol Boat and the ships have different lengths, different counts and different labels. The differences are stated in the table below.

Type of ships	Length of Ships	Counts of ships	Labels
Carrier	5 squares	1	CCCCC
Battleship	4 squares	2	BBBB
Destroyer	3 square	1	DDD
Submarine	3 squares	1	SSS
Patrol Boat	2 squares	4	PP

After players clarify the places of their own. Each player will try to guess where other player's ships are and for that they made a new 10x10 board of squares .Then they make their move by specifying the square which they want to play .For example if Player1 says 1,E square and if that place has a ship in Player2's own board the place will be marked as "X" but if it doesn't have a ship then it will be marked as "O".If one ship is sunk down , players should also put a "X" beside of the ships that have been sunked.For example if Player2's ships are sunked then the Player1 will win. In this program players put their ship's places and the moves that they will play and the program will show the moves and the winner at final position with both of their tables.

DESIGN

THE PROBLEM AND ITS SOLUTION

The program should read 4 files definitely and 2 files optionally then the program should make a new file called Battleship.out and make program write each move.The obligatory files are the files where each players put their ship's places, there are one file for each player and other 2 obligatory files consist of each player's moves . I used the 2 optional files too which consist of places of Battleship and Patrol Boat ships places since they have more than one count, there have been some complex situations so I decided to use these files.Also the program should pick the errors too.At first it needs to find out if the text file number is 4 and if the files in the directory or not. Also the program should pick the wrong moves and write the errors into the output file and keep reading.The wanted move type is "the row number, the column number" if there are less than one information the program will write IndexError , if the values are like "5,5" or "E,E" or "5,2,E" it will write ValueError, if the number or the letter is out of range like "5,K" or "20,A" it will write Assertion Error.If there are any other exceptions it will write "kaBOOM:run for your life". If there are no errors the output will have round number, player move, grid size, each player's playboard and ship names as stated below

Player1's Move

Round: 1 Grid Size: 10x10

Player1's Hidden Board		Player2's Hidden Board	
A B C D E F G H I J		A B C D E F G H I J	
1 - - - - - - - - - -		1- - - - - - - - - -	
2 - - - - - - - - - -		2- - - - - - - - - -	
3 - - - - - - - - - -		3- - - - - - - - - -	
4 - - - - - - - - - -		4- - - - - - - - - -	
5 - - - - - - - - - -		5- - - - - - - - - -	
6 - - - - - - - - - -		6- - - - - - - - - -	
7 - - - - - - - - - -		7- - - - - - - - - -	
8 - - - - - - - - - -		8- - - - - - - - - -	
9 - - - - - - - - - -		9- - - - - - - - - -	
10- - - - - - - - - -		10- - - - - - - - - -	
Carrier	-	Carrier	-
Battleship	--	Battleship	--
Destroyer	-	Destroyer	-
Submarine	-	Submarine	-
Patrol Boat	----	Patrol Boat	----

Enter your move:5,E

PROTOTYPE OF THE SOLUTION

I made 6 functions in this program which are create, errors battleship_1, battleship_2, tables and outputs.At first the program will read the files and if there is an error it will write the error and stops the program. If there is no error it will call create function and create function will make 4 different dictionary and 2 list then program will call errors function.Errors function will consider if there are any errors in players moves and it will write error and keep working on other moves so it doesn't stop when there is an error. If there is no error in a particular move it will find out whose move is it and if its Player1's move it will call battleship_1 function and in that function it will change the value of that move and then it will call tables function and the table function will write 2 of the table's current situations as stated above. If the move is Player2's move it will call battleship_2 function and this function will change the value of that move and then it will also call tables function and tables function will write 2 of the table's current situations as stated above.This will keep moving until one player's ships are sunked down if so it will write which player wins and the final information into output file .If Player2 will find all of Player1's ships the output will be "Player2 wins!!!" and final information tables. If Player1 will find all of Player2's ships, program will also consider Player2's last move because round should be complete and after that move, if Player2 also finds all of Player1's ships the output will be "It's a Draw" and the final information tables but if Player2 doesn't find all of Player1's ships the output will be "Player1 wins!!!" and final information tables.

THE DATA STRUCTURES USED IN PROGRAM AND THEIR PURPOSES

I used 6 dictionary and 2 of them are nested dictionaries. One of the nested dictionary name is place_board and it is considered as the playboards of players so at first all of place's values were "-", after making moves the values turnes into "X" or "O". Other nested dictionary name is with_letters, this dictionary has the actual places with letters of ships for example if G1,G2,G3 places are named as "D" it means in those squares there is a Destroyer ship and in battleship_1, battleship_2 functions will check if those places have ships or not according to this dictionary. Other 4 dictionaries were for calculating if a complete ship is sunked or not, 2 of their names are table_1 and table_2 and they have ship's first letters as key and their values as "-" for each player.The other 2 dictionaries names are b_p1 and b_p2, they have only ship places as key and first letters of ships as values so battleship_1 and battleship_2 will look for if ships are sunked down in b_p1 and b_p2, it will change "-" values in table_1 and table_2 and make them "X".For example after Player1's move if Carrier ship is sunked down the output will have "Carrier X ".An example is stated below

Carrier	-	Carrier	-
Battleship	--	Battleship	--
Destroyer	-	Destroyer	-
Submarine	-	Submarine	-
Patrol Boat	----	Patrol Boat	----

Enter your move:5,E

After this move a battleship is sunked down

Carrier	X	Carrier	-
Battleship	--	Battleship	--
Destroyer	-	Destroyer	-
Submarine	-	Submarine	-
Patrol Boat	----	Patrol Boat	----

Also I used lists and sets. There are lists for the 4 input files they have all of the lines.For the files that has players moves there are 3 lists one of them is has Player1’s moves seperately other one has Player2’s moves seperately and the third one is made by using “zip” command of these 2 lists. Also I made 2 sets named set_1 and set_2 . They are consist of table_1 and table_2 ’s values as sets and after that I turned these sets into lists.

PROGRAMMER'S CATALOGUE

```
import sys
import string
words = string.ascii_uppercase[0:10]
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
with_letters = dict()
place_board = dict()
table_1 = {"C": "-", "B1": "-", "B2": "-", "D": "-", "S": "-", "P1": "-", "P2": "-", "P3": "-", "P4": "-"}
table_2 = {"C": "-", "B1": "-", "B2": "-", "D": "-", "S": "-", "P1": "-", "P2": "-", "P3": "-", "P4": "-"}
```

```
optional_1 = open("OptionalPlayer1.txt", "r")
optional_2 = open("OptionalPlayer2.txt", "r")
output_f = open("Battleship.out", "w")
output_f.close()
output_f = open("Battleship.out", "a")
options_1 = optional_1.readlines()
options_2 = optional_2.readlines()
try:  # if there are less than 4 files it will print IndexError
    f = sys.argv[1]
    f2 = sys.argv[2]
    f1_in = sys.argv[3]
    f2_in = sys.argv[4]
    try:
        # if there is a wrong input file is written in terminal, this will consider all the combinations of wrong input
        # file names and will print the wrong files with their names
        file = open(f, "r")
        try:
            file2 = open(f2, "r")
            try:
                file1_in = open(f1_in, "r")
                try:
                    file2_in = open(f2_in, "r")
                except FileNotFoundError:
                    outputs("IOError: input file {} is not reachable.".format(f2_in))
            except FileNotFoundError:
                try:
                    file2_in = open(f2_in, "r")
                    outputs("IOError: input files {} are not reachable".format(f1_in))
                except FileNotFoundError:
                    outputs("IOError: input files {} {} are not reachable".format(f1_in, f2_in))
```

```
    except FileNotFoundError:
        try:
            file1_in = open(f1_in, "r")
            try:
                file2_in = open(f2_in, "r")
                outputs("IOError: input file {} is not reachable".format(f2))
            except FileNotFoundError:
                outputs("IOError: input files {} {} are not reachable".format(f2, f2_in))
        except FileNotFoundError:
            try:
                file2_in = open(f2_in, "r")
                outputs("IOError: input files {} {} are not reachable".format(f2, f1_in))
            except FileNotFoundError:
                outputs("IOError: input files {} {} {} are not reachable".format(f2, f1_in, f2_in))
```

```
except FileNotFoundError:
    try:
        file2 = open(f2, "r")
        try:
            file1_in = open(f1_in, "r")
            try:
                file2_in = open(f2_in, "r")
                outputs("IOError: input file {} is not reachable.".format(f))
            except FileNotFoundError:
                outputs("IOError: input files {} {} are not reachable.".format(f, f2_in))
        except FileNotFoundError:
            try:
                file2_in = open(f2_in, "r")
                outputs("IOError: input files {} {} are not reachable.".format(f, f1_in))
            except FileNotFoundError:
                outputs("IOError: input files {} {} {} are not reachable.".format(f, f1_in, f2_in))
    except FileNotFoundError:
        try:
            file1_in = open(f1_in, "r")
            try:
                file2_in = open(f2_in, "r")
                outputs("IOError: input files {} {} are not reachable.".format(f, f2))
            except FileNotFoundError:
                outputs("IOError: input files {} {} {} are not reachable.".format(f, f2, f2_in))
        except FileNotFoundError:
            try:
                file2_in = open(f2_in, "r")
                outputs("IOError: input files {} {} {}are not reachable.".format(f, f2, f1_in))
            except FileNotFoundError:
                outputs("IOError: input files {} {} {} {} are not reachable.".format(f, f2, f1_in, f2_in))
```

```
except IndexError:
    outputs("IndexError: number of inputs less than expected")
except Exception:
    outputs("kaBOOM: run for your life.")
else:
    try:
        player_txt = file.readlines()
        player2_txt = file2.readlines()
        player1_in = file1_in.readlines()
        player2_in = file2_in.readlines()
        one_list = (player1_in[0].strip()).split(";")
        two_list = (player2_in[0].strip()).split(";")
        one_in_list = [one_list[i].split(",") for i in range(len(one_list) - 1)]
        two_in_list = [two_list[i].split(",") for i in range(len(two_list) - 1)]
        all_in_list = list(zip(one_in_list, two_in_list))
        create()
        errors()
    except NameError:
        pass
    except Exception:
        outputs("kaBOOM: run for your life.")
```

```
def create():
    global b_p1, b_p2  # these dictionaries will have only ships places
    b_p1 = dict()
    b_p2 = dict()
    with_letters["Player1"] = dict()  # this dictionary will have blanks and ships places together
    with_letters["Player2"] = dict()
    place_board["Player1"] = dict()  # this will only have '-' and change during the play
    place_board["Player2"] = dict()
    for word in ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]:
        for number in range(1, 11):
            place_board["Player1"][word + str(number)] = "-"
            place_board["Player2"][word + str(number)] = "-"
    for word in ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]:
        for number in range(1, 11):
            with_letters["Player1"][word + str(number)] = "-"
            with_letters["Player2"][word + str(number)] = "-"
```

```
for lines in player_txt:  # this will add carrier, submarine and destroyer ships places into with_letters[Player1]
    player1 = (lines.strip()).split(";")
    for i in ["C", "S", "D"]:
        if i in player1:
            nindex = player1.index(i)
            for n in range(nindex, nindex + player1.count(i)):
                if player_txt.count(lines) == 1:
                    with_letters["Player1"][words[n] + str(player_txt.index(lines) + 1)] = i
                else:
                    with_letters["Player1"][words[n] + str(player_txt.index(lines) + 1)] = i
                    with_letters["Player1"][words[n] + str(player_txt.index(lines,
                                                                            player_txt.index(lines) + 1) + 1)] = i
for lines in player2_txt:   # this will add carrier, submarine and destroyer ships places into with_letters[Player2
    player2 = (lines.strip()).split(";")
    for i in ["C", "S", "D"]:
        if i in player2:
            nindex = player2.index(i)
            for n in range(nindex, nindex + player2.count(i)):
                unknown = with_letters["Player2"][words[n] + str(player2_txt.index(lines) + 1)]
                if player2_txt.count(lines) == 1 and unknown == "-":
                    with_letters["Player2"][words[n] + str(player2_txt.index(lines) + 1)] = i
                elif player2_txt.count(lines) > 1 and unknown == "-":
                    with_letters["Player2"][words[n] + str(player2_txt.index(lines) + 1)] = i
                    unknow = with_letters["Player2"][words[n] + str(player2_txt.index(lines,
                                                                                      player2_txt.index(lines) + 1) + 1)]
                    if unknow == "-":
                        with_letters["Player2"][words[n] + str(player2_txt.index(lines,
                                                                                 player2_txt.index(lines) + 1) + 1)] = i
```

```
    key_1 = [key for key, value in with_letters["Player1"].items() if value in ("C", "S", "D")]
    key_2 = [key for key, value in with_letters["Player2"].items() if value in ("C", "S", "D")]
    value_1 = []
    value_2 = []
    for k in key_1:
        value_1.append(with_letters["Player1"].get(k))
    b_p1 = dict(zip(key_1, value_1))
    for k in key_2:
        value_2.append(with_letters["Player2"].get(k))
    b_p2 = dict(zip(key_2, value_2))
    for line in options_1:  # this will add battleship and patrol boat ships places into with_letters[Player1]
        line = (line.strip()).split(",")
        if line[1][2] == "r" and line[0][0] == "B":
            for i in range(words.index(line[1][0]), words.index(line[1][0]) + 4):
                b_p1[words[i] + str(line[0][3:])] = line[0][0:2]
                with_letters["Player1"][words[i] + str(line[0][3:])] = line[0][0]
        elif line[1][2] == "d" and line[0][0] == "B":
            for i in range(int(line[0][3:]), int(line[0][3:]) + 4):
                b_p1[line[1][0] + str(i)] = line[0][0:2]
                with_letters["Player1"][line[1][0] + str(i)] = line[0][0]
        elif line[1][2] == "r" and line[0][0] == "P":
            for i in range(words.index(line[1][0]), words.index(line[1][0]) + 2):
                b_p1[words[i] + str(line[0][3:])] = line[0][0:2]
                with_letters["Player1"][words[i] + str(line[0][3:])] = line[0][0]
        elif line[1][2] == "d" and line[0][0] == "P":
            for i in range(int(line[0][3:]), int(line[0][3:]) + 2):
                b_p1[line[1][0] + str(i)] = line[0][0:2]
                with_letters["Player1"][line[1][0] + str(i)] = line[0][0]
```

```
    for line in options_2:  # this will add battleship and patrol boat ships places into with_letters[Player2]
        line = (line.strip()).split(",")
        if line[1][2] == "r" and line[0][0] == "B":
            for i in range(words.index(line[1][0]), words.index(line[1][0]) + 4):
                b_p2[words[i] + str(line[0][3:])] = line[0][0:2]
                with_letters["Player2"][words[i] + str(line[0][3:])] = line[0][0]
        elif line[1][2] == "d" and line[0][0] == "B":
            for i in range(int(line[0][3:]), int(line[0][3:]) + 4):
                b_p2[line[1][0] + str(i)] = line[0][0:2]
                with_letters["Player2"][line[1][0] + str(i)] = line[0][0]
        elif line[1][2] == "r" and line[0][0] == "P":
            for i in range(words.index(line[1][0]), words.index(line[1][0]) + 2):
                b_p2[words[i] + str(line[0][3:])] = line[0][0:2]
                with_letters["Player2"][words[i] + str(line[0][3:])] = line[0][0]
        elif line[1][2] == "d" and line[0][0] == "P":
            for i in range(int(line[0][3:]), int(line[0][3:]) + 2):
                b_p2[line[1][0] + str(i)] = line[0][0:2]
                with_letters["Player2"][line[1][0] + str(i)] = line[0][0]
```

```
def errors():
    number = [int(i) for i in range(0, 400)]
    alphabet = string.ascii_uppercase
    same_ones = []
    outputs("Battle of Ships Game\n\n")
    for j in range(len(all_in_list)):
        for i in range(len(all_in_list[j])):  # this two loop will take moves one by one
            try:  # if there is any type of error it will print the error and keep moving for following move
                places = all_in_list[j][i]
                if len(places) < 2 or places[0] == '' or places[1] == '':
                    raise IndexError
                elif len(places) > 2 or int(places[0]) not in number or places[1] not in alphabet:
                    raise ValueError
```

```
                elif len(places) == 2:
                    assert 0 < int(places[0]) < 11 and places[1] in alphabet[0:10]
                    if all_in_list[j].count(places) == 2:  # if both of the players made the same move at the same time
                        same_ones.append(places)
                        if len(same_ones) % 2 == 1:  # if it is Player1 move it will call battleship_1
                            battleship_1(places)
                        elif len(same_ones) % 2 == 0:  # if it is Player2 move it will call battleship_1
                            battleship_2(places)
                            if len(list_2) == 1 and list_2[0] == "X" and len(list_1) == 1 and list_1[0] == "X":
                                outputs("It is a Draw!\n\nFinal Information\n\n")
                                tables()
                                break
                            elif len(list_1) == 1 and list_1[0] == "X":
                                outputs("Player2wins\n\nFinal Information\n\n")
                                tables()
                                break
                            elif len(list_2) == 1 and list_2[0] == "X":
                                outputs("Player1 Wins\n\nFinal Information\n\n")
                                tables()
                                break
                    else:
                        if places == all_in_list[j][0]:  # if it is Player1 move it will call battleship_1
                            battleship_1(places)
                        elif places == all_in_list[j][1]:  # if it is Player2 move it will call battleship_2
                            battleship_2(places)
```

```
                    if len(list_2) == 1 and list_2[0] == "X" and len(list_1) == 1 and list_1[0] == "X":
                        outputs("It is a Draw!\n\nFinal Information\n\n")
                        tables()
                        break
                    elif len(list_1) == 1 and list_1[0] == "X":
                        outputs("Player2 Wins!\n\nFinal Information\n\n")
                        tables()
                        break
                    elif len(list_2) == 1 and list_2[0] == "X":
                        outputs("Player1 Wins!\n\nFinal Information\n\n")
                        tables()
                        break
            except IndexError:
                outputs("IndexError:The place you entered is not exist.\n")
            except ValueError:
                outputs("ValueError:The place you entered is not valid\n")
            except AssertionError:
                outputs("AssertionError: Invalid Operation\n")
            except Exception:
                pass
```

This function will call battleship_1 and battleship_2 one by one

```
def battleship_1(x):
    output_f.write("Player1's Move\n\nRound: {}\t\t\t\t\tGrid Size: 10x10\n\n".format(str((one_in_list.index(x)) + 1)))
    print("Player1's Move\n\nRound: {}\t\t\t\t\tGrid Size: 10x10\n".format(str((one_in_list.index(x)) + 1)))
    tables()
    output_f.write("Enter your move:{}\n\n".format(x[0] + "," + x[1]))
    print("Enter your move:{}\n".format(x[0] + "," + x[1]))
    if with_letters["Player2"][x[1] + str(x[0])] != "-":
        place_board["Player2"][x[1] + str(x[0])] = "X"  # this will change "-" to "X" if the place is full
        b_p2[x[1] + str(x[0])] = "X"
    else:
        place_board["Player2"][x[1] + str(x[0])] = "O"  # this will change "-" to "O" if the place is NOT full
    for i in ["C", "B1", "B2", "D", "S", "P1", "P2", "P3", "P4"]:
        if i not in b_p2.values():
            table_2[i] = "X"
    global set_2, list_2
    set_2 = set(table_2.values())
    list_2 = list(set_2)
    if len(list_2) == 1 and list_2[0] == "X":
        finde = one_in_list.index(x)
        battleship_2(two_in_list[finde])
```

```
def battleship_2(x):
    output_f.write("Player2's Move\n\nRound: {}\t\t\t\t\tGrid Size: 10x10\n\n".format(str(two_in_list.index(x) + 1)))
    print("Player2's Move\n\nRound: {}\t\t\t\t\tGrid Size: 10x10\n\n".format(str(two_in_list.index(x) + 1)))
    tables()
    output_f.write("Enter your move:{}\n\n".format(x[0] + "," + x[1]))
    print("Enter your move:{}\n".format(x[0] + "," + x[1]))
    if with_letters["Player1"][x[1] + str(x[0])] != "-":
        place_board["Player1"][x[1] + str(x[0])] = "X"  # this will change "-" to "X" if the place is full
        b_p1[x[1] + str(x[0])] = "X"
    else:
        place_board["Player1"][x[1] + str(x[0])] = "O"  # this will change "-" to "O" if the place is NOT full
    for i in ["C", "B1", "B2", "D", "S", "P1", "P2", "P3", "P4"]:
        if i not in b_p1.values():
            table_1[i] = "X"
    global set_1, list_1
    set_1 = set(table_1.values())
    list_1 = list(set_1)
```

battleship_1 and battleship_2 will call tables function

```
def tables():  # this function will call outputs function for printing all the tables into output file
    table_str = ""
    table_str += "Player1's Hidden Board\t\tPlayer2's Hidden Board\n  A B C D E F G H I J\t\t  A B C D E F G H I J\n"
    plist = [table_1['P1'], table_1['P2'], table_1['P3'], table_1['P4']]
    plist_2 = [table_2['P1'], table_2['P2'], table_2['P3'], table_2['P4']]
    blist = [table_1['B1'], table_1['B2']]
    blist_2 = [table_2['B1'], table_2['B2']]
    for i in plist, plist_2, blist, blist_2:
        i.sort()
        i.reverse()
    for i in numbers:
        if i < 10:
            table_str += "{} ".format(i)
            for j in range(len(words)):
                table_str += "{} ".format(place_board["Player1"][words[j] + str(i)])
            table_str += "\t\t{}".format(i)
            for w in range(len(words)):
                table_str += "{} ".format(place_board["Player2"][words[w] + str(i)])
            table_str += "\n"
        else:
            table_str += "{}".format(i)
            for m in range(len(words)):
                table_str += "{} ".format(place_board["Player1"][words[m] + str(i)])
            table_str += "\t\t{}".format(i)
            for n in range(len(words)):
                table_str += "{} ".format(place_board["Player2"][words[n] + str(i)])
            table_str += "\n\n"
```

```
table_str += "Carrier \t{}\t\tCarrier \t{}\nBattleship\t{}{}\t\tBattleship\t{}{}\nDestroyer\t{}\t\tDestroyer" \
"\t{}\n".format(table_1['C'], table_2['C'], blist[0], blist[1], blist_2[0], blist_2[1], table_1['D'], table_2['D'])
table_str += "Submarine\t{}\t\tSubmarine\t{}\nPatrol Boat\t{}{}{}{}\t\tPatrol Boat\t{}{}{}{}\n\n".format\
(table_1['S'], table_2['S'], plist[0], plist[1], plist[2], plist[3], plist_2[0], plist_2[1], plist_2[2], plist_2[3])
outputs(table_str)
```

errors, battleship_1 and battleship_2 functions call the outputs function for printing all of the tables into Battleship.out file and command line

```
def outputs(x):
    print(x)
    output_f.write(x)
```

USER CATALOGUE

The user should make 6 files.First file should clarify the places of Player1's ships.Second file should clarify the places of Player2's ships.Third file should have the moves that Player1 will play in order to "row number, column letter" and the places should be seperated by ";".4th file should have the moves that Player2 will play in order to "row number, column letter" and the places should be seperated by ";".5th file has Player1's battleships and patrol boats places one by one. 6th file has Player2's battleships and patrol boats places one by one.

GRADING

Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5.
Using Meaningful Naming	5	4.
Using Explanatory Comments	5	3.
Efficiency (avoiding unnecessary actions)	5	5.
Function Usage	25	25
Correctness	35	35
Report	20	20
There are several negative evaluations	...	...