ASSIGNMENT REPORT

BBM103

ASSIGNMENT-4

EMRE ÇİĞİL – 2210356033

03.01.2023

ANALYSIS

In this problem, they ask us to make a game and we call this game 'Battleship'. The goal of the game is to shoot down the opponent's ships and the ship sinks when all the parts of the ship are hit. The player whose entire ship sinks loses the game

We know that there is more than one type of ship in the game, and we will determine different sizes and numbers for each ship. In the game, moves are made in order and we need to attack a point that is in the coordinate system. otherwise, it will be indicated that it is the wrong coordinate and will lose its order.

When the game is over, it writes down the winning player or writes a draw. The latest final information is given and the final version of the ships is shown.

The project should have certain rules. You have to make the right moves, or it will fail. You should not use the wrong entry. You must make sure that you have opened the correct files, or it will give the error. We need to do the project by considering these possibilities.

DESIGN

I used 9 functions in total. The names of the function are: create_matrix, mat_1, ships, create_move_list, attack1, attack2, game, input and output.

The 'create_matrix' function was created to place ships in the coordinate plane. With this function, the location of the ships is determined secretly in the coordinate plane.

The 'mat_1' function creates a coordinate plane for players to see, and players make their moves by looking at this coordinate plane.

The 'ships' function detects the parts of the ships and makes an entire ship, and we identify the sinking ships with this function.

The 'create_move_list' function reads input files and adds players' moves to a list. We use this list in subsequent functions. In turn, we take the moves from this list.

The 'attack1' and 'attack2' function bombard the location in the coordinate plane according to the players' moves. There is an attack function for each player.

The 'game' function is the function that plays the game. All functions run in the background but the 'game' function outputs to the screen.

The 'input' and 'output' functions respectively receive information from the user and output it to the user after the information received is interpreted.

In the output, the matrices are written side by side and there is a scoreboard just below it. Moves are written and if a part of the ship is hit, the '-' character in the matrix changes to the 'X' character. And above the matrix is written how many rounds it is in.

I used many functions while designing the project. Everything has become simpler because I use many functions. and I didn't have to print out every function. I printed out a function using functions in one another. I created lists with various tasks so that I simplified the operation of the project. Some lists are inside the ships, some lists are in the moves, some lists are in the leaderboard, some lists are created to make a matrix. Some lists were created for the purpose of checking.

I tried to minimize errors by using 'try-exception'. When opening files there may be errors, wrong moves can be made, ships can be positioned incorrectly, etc.

I tried using a dictionary while doing the project. I used a dictionary in the previous project and felt it was productive. In this project, I did not get the efficiency I expected from dictionaries. That's why I made use of lists. The lists in my project are a list of ships, a list that gives us the coordinates of the ships, attack lists, checklists, and matrix lists.

PROGRAMMER'S CATALOGUE

It took me time to solve the problem, but I had to be fast. I drafted what I needed to do before I wrote the codes. I thought about what functions I should create. I thought about the function of functions and started writing. I made mistakes many times in the output, but I continued without giving up. I researched new methods.

It took 1 day to design. It took almost 4 to 5 days to write down the functions. It took 1 day to write a report.

```
import sys
try:
    arg1, arg2, arg3, arg4 = sys.argv[1], sys.argv[2], sys.argv[3],
sys.argv[4]
except SystemError:
    print("Error: There ara more or less argument than expected.")
file_output = open("Battleship.out", "w", encoding="UTF-8")
try:
    p1_ship = open(arg1, "r", encoding="UTF-8")
except IOError:
    print("IOError: input file {} is not reachable.".format("Player1.txt"))
try:
    p2_ship = open(arg2, "r", encoding="UTF-8")
except IOError:
    print("IOError: input file {} is not reachable.".format("Player2.txt")
```

Here I used the 'open' function to open the necessary input files. If the file is incorrect, I indicated this with 'try-except'.

I imported sys to open input files. I determined how many arguments there were and assigned them to different values.

```
def create_matrix():
    try:
        p1_mat = open(arg1, "r", encoding="UTF-8")
    except IOError:
        print("IOError: input file(s) {} is/are not
reachable.".format("Player1.txt"))
    try:
        p2_mat = open(arg2, "r", encoding="UTF-8")
    except:
        print("IOError: input file(s) {} is/are not
reachable.".format("Player2.txt"))
    for i in p1_mat:
        i = i.strip("\n")
        i = i.split(";")
        for k in range(len(i)):
            i[k] = "-"
        martix1.append(i)
    for i in p2_mat:
        i = i.strip("\n")
        i = i.split(";")
        for k in range(len(i)):
            i[k] = "-"
        martix2.append(i)
   p1_mat.close(), p2_mat.close()
create_matrix()
```

With this function we read the input files and create a matrix board according to the files. This matrix we have created is made for players. Then I closed the input files. I used the ‘for’ loop to create matrix.

```
def mat_1(x):  # bu fonksiyon oluşturulan matriks listesini ve batan gemi
tablosunu outputta göstermek için yazıldı.
    if x == 1:
        print("Player1's Board", end="             ", file=file_output),
print("Player2's Board", file=file_output)
        for i in range(len(chart1)):
            for k in range(len(chart1[i])):
                if chart1[i][k] != "" and martix1[i][k] != "X":
                    martix1[i][k] = chart1[i][k]                #bu döngüyü
final durumunu göstermek için kullandım.
        for i in range(len(chart2)):
            for k in range(len(chart2[i])):
                if chart2[i][k] != "" and martix2[i][k] != "X":
                    martix2[i][k] = chart2[i][k]
    else:
        print("Player1's Hidden Board", end="      ", file=file_output),
print("Player2's Hidden Board", file=file_output)
    print(" ", " ".join(alphabet[:len(martix1)]), end="       ",
file=file_output), print(" ", " ".join(alphabet[:len(martix1)]),
file=file_output)
    y = 1
    global list_ship1                #bu for döngüsü raundlar içi kullandım.
    global list_ship2
    for i in range(len(martix1)):
        if y < 10:
            print(str(y), " ".join(martix1[i]), end="       ",
file=file_output), print(str(y), " ".join(martix2[i]), file=file_output)
            y += 1
        else:
```

```
            print(str(y) + " ".join(martix1[i]), end="       ",
file=file_output), print(str(y) + " ".join(martix2[i]), file=file_output)
            y += 1
    print("\nCarrier     ", *list_ship1[0], 12*" ", "Carrier     ",
*list_ship2[0], "\nBattleship  ", *list_ship1[1], 10*" ", "Battleship  ",
*list_ship2[1],
          "\nDestroyer   ", *list_ship1[2], 12*" ", "Destroyer   ",
*list_ship2[2], "\nSubmarine   ", *list_ship1[3], 12*" ", "Submarine   ",
          *list_ship2[3], "\nPatrol Boat ", *list_ship1[4], 6*" ", "Patrol
Boat ", *list_ship2[4], "\n", file=file_output)
```

I printed the characters in the matrix side by side using a 'for' loop. I printed out the final status or normal rounds using 'if' and 'else'.

I printed out the ships on the leaderboard and called up the ship lists I had created to show what the score was.

I used various commands and methods in this function. I understood that the 'join' method is very functional in this function.

I made the function more efficient by using a method I discovered after my research to get printouts.

I can use other functions by making ship lists global.

```
battleship1, carrier1, destroyer1, submarine1, patrol1 = [], [], [], [], []
battleship2, carrier2, destroyer2, submarine2, patrol2 = [], [], [], [], []
def ships():
```

Since this function is too long, I will not be able to show them all here. With the 'for' loop and 'if-else' methods I used in this function, I checked the position of the ships vertically and horizontally and added them to the lists I showed above. I created a list specific to each ship and made my work easier among other functions. The reason why this function was long was because I thought that the number of ships that had already been given to us could change. But since the number of ships would be fixed, I could have written this more briefly.

Inside the loop there are conditions according to the letters. When these conditions are met, the necessary actions are taken, and the latest ships are added to the lists. I used new conditions within the conditions, considering the possibility of the ships being next to each other. Thanks to these circumstances, I tried to secure my project.

```
def attack1(a):           #Bu fonksiyon saldırı noktalarını eklediğimiz
listeleri okuyup o noktada gemi olup olmamasını kontrol ediyor ve ona göre
çıktı çıkmasını sağlıyor.
    try:
        assert int(player1move[a][0:-2]) <= 10 and player1move[a][-1] in
alphabet, "AssertionError: Invalid Operation."
        try:
            coor_hor1, coor_ver1, e = (int(player1move[a][0:-2]) - 1),
alphabet.index(player1move[a][-1]), player1move[a]
            if chart2[coor_hor1][coor_ver1] == "":
                martix2[coor_hor1][coor_ver1] = "O"
            elif chart2[coor_hor1][coor_ver1] == "B":
                martix2[coor_hor1][coor_ver1] = "X"
                for i in range(len(battleship2)):
                    if e in battleship2[i]:
                        b1.append(i)
                        if b1.count(0) == 4 or b1.count(1) == 4:
                            list_ship2[1].pop(), list_ship2[1].insert(0,
"X")
                            b1.remove(i)
            elif chart2[coor_hor1][coor_ver1] == "C":
                martix2[coor_hor1][coor_ver1] = "X"                      #bu
döngülerle gemilerin olduğu listede harf taraması yapıp saldırı noktasıyla
aynı
                for i in range(len(carrier2)):                           #
indekstelerse tablodaki - yi X yapar ve gemi sayacı listesine vurulan
parçayı ekler
                    if e in carrier2[i]:                                  #
bütün parça vurulduysa gemi batar ve sayaç bunu gösterir
                        c1.append(i)
                if c1.count(0) == 5:
                    list_ship2[0].pop(), list_ship2[0].insert(0, "X")
            elif chart2[coor_hor1][coor_ver1] == "D":
                martix2[coor_hor1][coor_ver1] = "X"
                for i in range(len(destroyer2)):
                    if e in destroyer2[i]:
                        d1.append(i)
                if d1.count(0) == 3:
                    list_ship2[2].pop(), list_ship2[2].insert(0, "X")
            elif chart2[coor_hor1][coor_ver1] == "S":
                martix2[coor_hor1][coor_ver1] = "X"
                for i in range(len(submarine2)):
                    if e in submarine2[i]:
                        s1.append(i)
                if s1.count(0) == 3:
                    list_ship2[3].pop(), list_ship2[3].insert(0, "X")
            elif chart2[coor_hor1][coor_ver1] == "P":
                martix2[coor_hor1][coor_ver1] = "X"
                for i in range(len(patrol2)):
                    if e in patrol2[i]:
                        p1.append(i)
                        if p1.count(0) == 2 or p1.count(1) == 2 or
p1.count(2) == 2 or p1.count(3) == 2:
                            list_ship2[4].pop(), list_ship2[4].insert(0,
"X"), p1.remove(i)
        except ValueError:
            print("ValueError: Wrong Value is entered!"),
print("ValueError: Wrong Value is entered!", file=file_output)
    except AssertionError as msg:
        print(msg,file=file_output), print(msg)
```

In this function, I read the players' moves from the input file and determined the location of the moves in the coordinate plane. If there is no ship at the specified point in the coordinate plane, I printed 'O'. I printed an 'X' if there was a ship at the designated point in the coordinate plane.

I identified the type of ship and added the ships to a special checklist accordingly and checked all the parts of the ships with the 'count' function. If all the pieces were hit, I added them to the scoreboard. I checked for incorrect entries with the phrase 'try - except' and printed it on the screen if there was a faulty function. If there is invalid input using 'Assertion Error', I printed it on the screen. There are two of these functions, and they are both similar. The only difference is that different lists and different input files are checked for each player and moves are made accordingly.

```
def game():   # Bu fonkiyon bu işlemlerin hepsini oyuna döker ve oyunu
çalıştırır
    y, x = 0, 1
    print("Battle of Ships Game", "\n", file=file_output)
    for i in range(max(len(player1move), len(player2move))+1):
#for döngüsüyle oyunun uzunluğu ve hamle sırası kontrol altına alınır.
        if "-" not in (''.join(map(''.join, list_ship1))):
            print("Player2 Wins!", "\n", file=file_output), print("Final
Information", "\n", file=file_output)
            mat_1(1)
# Gerekli fonksiyonlar burada çağırılır
            break
        elif "-" not in (''.join(map(''.join, list_ship2))):
            print("Player1 Wins!", "\n", file=file_output), print("Final
Information", "\n", file=file_output)
            mat_1(1)
            break
        elif i == max(len(player1move), len(player2move)):
            print("It is a Draw!", "\n", file=file_output), print("Final
Information", "\n", file=file_output)
            mat_1(1)
            break
        print("Player1's Move", "\n", file=file_output), print("Round :",
x, end="                    ", file=file_output), print("Grid Size:",
str(len(martix1))+"x"+str(len(martix1)), "\n", file=file_output)
        mat_1(0)
        print("Enter your move: ",player1move[y], "\n", file=file_output),
attack1(y)
        print("Player2's Move", "\n", file=file_output), print("Round :",
x, end="                    ", file=file_output), print("Grid Size:",
str(len(martix1))+"x"+str(len(martix1)), "\n", file=file_output)
```

```
        mat_1(0)
        print("Enter your move: ", player2move[y], "\n", file=file_output),
attack2(y)
        x += 1
        y += 1
game()
```

With this function, the game is played and writes some sentences that should appear in the output. I called the 'mat_1' function to show the matrix and the scoreboard inside this function because we need to show them in each round. We find out if the game is finished or not using a for loop. I determine who won the game by the 'if-else' conditions I used. And after the game is over, I call the 'mat_1' function for the last time, and this time it shows the final status. It shows which ship of the players did not sink in the finale. In this function, I show the letter x to show which round it is in. Using the letter y, I print the next move of the players on the screen. I print the grid size on the screen using the length of the matrix.

USER CATALOGUE

It is very simple for users to play this game. All you need to do is select a point in the coordinate plane. If there is a ship at this point of your choice, the system understands it and sinks the part of the ship at that point. You see on the screen who won last or drew last.

This game is normally played by taking individual inputs one by one, but to play faster, we have an input file and all the moves are in this file. The reason this is so is to be able to quickly check whether this project is currently running the game.

When you make a wrong move, the system notices it and prints on the screen that you made the wrong move. Because you typed the wrong move, the game moves to the next player. if you hit the same place twice, you will lose your move.

To briefly mention ships, there are 5 types of ships. There is 1 of the carrier ship, and its length is 5 units. There are 2 of the battle ships, and its length is 4 units. There is 1 of the destroyer ship, and its length is 3 units. There is 1 of the submarine vessel, and its length is 3 units. There are 4 of the oil vessels, and their length is 2 units.

In the game you have to shoot down all the parts of the ship to sink a ship. The hit part of the ship will appear as an 'X' on the grille. Thanks to this, sinking the ship will be easier. You can track how many ships have sunk in the leaderboard below the grids. And you can see how many rounds you are in at the top right of the grids. You can see which player is in the top left of the grids. You can look at the bottom of the leaderboard to see the move.

Evaluation	Points	Evaluate Yourself / Guess Grading
Readable Codes and Meaningful Naming	5	4

Evaluation	Points	Evaluate Yourself / Guess Grading
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	4
Function Usage	15	15
Correctness, File I/O	30	30
Exceptions	20	18
Report	20	17
There are several negative evaluations	…	…

Reference:

- https://stackoverflow.com/
- https://www.geeksforgeeks.org/
- https://www.w3schools.com/
- https://www.reitix.com/
- https://www.pythontutorial.net/python-basics/python-read-text-file/