Hacettepe University Department of Computer Engineering

BBM 103 Assignment 4 Report

04.01.2023

Gülvera Yazılıtaş – 2210356111

Index

1 Analysis

The assignment is to create a program name “ Battleship Game ”. In this program data of players as moves and locations of their ships are wanted to make moves and deterime the winner of the game. The data are given in four txt file and the outputs of the program should be written in both battleship.out file and command line. Also progrram should see possible errors and warn users by printing a message.

The taken moves belongs two players should be implemented by order (first player 1 and second player2). Players should not see their places of ships during the game but program have to know where the ships are so that it can show the result of shots (if it is successful shot or not).

1.1 Data :

The data of a patient have six categories:

1. Player1.txt
2. Player2.txt
3. Player1.in
4. Player2.in

1.2 Possible errors when running the program;

- Most possible errors may occur when taking moves. For example if a move out of the format (number(1-10),letter(A-J)) program prints an error message about this move.
- In addition, opening files can give errors. For that, given files names should be controlled.

2 Design

- **Reading Inputs Problem:** We start to read and convert the data to proper form by usuing reading function. I used to List to hold the ships loacation and this lists are also used for printing game board. One of them is hidden list , which does not show the ships location and the other one is mainlist , which have exact indices of ships. Both players have these two lists at the beginning. I prefered to hold ships places in list data type instead of dictionary etc. Because every space have its unique index in 100 indices of list. It makes other operations easier.

- **Writing to output file**
  To write every message in the program generates a writing function can be made and used after every command execution. Understanding the commands
  A control mechanism , namely if - elif, controls the command variable and finds the proper operation for it. For example, if the command is 'create', the create code part should be executed.

- **Problem of locations of ships:**

  Reason of Problem: To find out which index will be shot and is there any piece of ship in that index , I had to know what are the exact indices of every ship.

  Solution: I designed an finding indices alghorithm. It works like this; firstly it takes list to check and item to find. For example , we want to find “carrier” ship in mainlist1 , to do that I preferd to use *enumerate* function. It reads every index in list and if it see letter ‘C’ , it appends that index to *list indices* and then returns the indices that it found. For instance;

  find_indices(list_to_check, item_to_find)=[22,23,24,25,26]

  With this method location of *Carrier, Submarine, Destroyer* can be found.

- **Problem of locations of ships: Battleship and Petrol Boat**

Game has total five types of ships. The Game has just one ship for threee types of ships. However “battleship” and “petrol boat” are in the inputs more than one. So locating true indices of each ship for these type of ships is a problem to write an alghorithm. (Because in this programm optional inputs do not used!)

Reason of Problem: When I created a mainlist (10x10) we have 2*4= 8 piece of battleships and 4*2=8 piece of petrol boats. When our *find_indices* function take their indices , they are not in an order in the list. For example,

find_indices(mainlist1, 'B') = [21,31,35,36,37,38,41,51]

Let say B1 has (21,31,41,51) and B2 has (35,36,37,38) indices. Our function uncapable of which index belongs to which ship so another alghorithm is needed in this part of problem.

Alghorithm of Solution:

1-) Finding B's: *Locating_b function* take a mainlist and B's indices(they are taken by *find _indices* function). Because of the ascending order of list of b indices ,

```
I take the first element of list as "a"

then in maintlist ,firstly, look for indices [a+1] and [a+2]

if they are both 'B'  then I check the [a+3] index
    if it is also 'B'

    then I determine first battleship indices as b1 = listofbindices[0:4]

    b2 = listofbindices[4:9]

elif mainlist[a+10] and mainlist[a+20]

    if mainlist[a+30] is also 'B'

            then b1 = [a,a+10,a+20,a+30] and b2 is all indices but the b1 has.
```

Then this function returns the two list b1 and b2. So we do not have any misloacation info about ships.

Alghorithm of Solution:

1-) Finding P's: *Locating_p function* take a mainlist and P's indices(they are taken by *find _indices* function). This alghoritm is more complicated then locating b's. First of all indices of p's have to been split in 4 pieces. Each pair has to have true indices of Petrol boats.

First I created a list called ' p_plerim' it holds 4 mini list that every of them has 2 indices.

For example, p_lerim = [[14,15],[28,29],[51,61],[88,98]]

Then I made a for loop that iterates 4 times(for 4 ships)

Inside the for loop I set 'a' to first element of list of p indices

Then rest of the alghoritm goes like this;

```
for i in range(4):
    a = pl__p_ind[0]
    if mainlist1forp[a+1] == 'P' and mainlist1forp[a+10] != 'P':
        p_lerim.append([a, a+1])
        pl__p_ind.remove(a)
        pl__p_ind.remove(a+1)
    elif mainlist1forp[a+1] != 'P' and mainlist1forp[a+10] == 'P':
        p_lerim.append([a, a + 10])
        pl__p_ind.remove(a)
        pl__p_ind.remove(a + 10)
    elif mainlist1forp[a+1] == 'P' and mainlist1forp[a+10] == 'P':  #
Critical elif
        if mainlist1forp[a+2] == 'P' and mainlist1forp[a+20] != 'P':
            p_lerim.append([a, a + 10])
            pl__p_ind.remove(a)
            pl__p_ind.remove(a + 10)
        elif mainlist1forp[a+2] != 'P' and mainlist1forp[a+20] == 'P':
            p_lerim.append([a, a + 1])
            pl__p_ind.remove(a)
            pl__p_ind.remove(a+1)
        elif mainlist1forp[a+2] == 'P' and mainlist1forp[a+20] == 'P':  #
Critical elif
            if mainlist1forp[a+3] == 'P' and mainlist1forp[a+30] != 'P':
                p_lerim.append([a, a + 1])
                pl__p_ind.remove(a)
                pl__p_ind.remove(a+1)
            elif mainlist1forp[a+3] != 'P' and mainlist1forp[a+30] == 'P':
                p_lerim.append([a, a + 10])
                pl__p_ind.remove(a)
                pl__p_ind.remove(a + 10)
            elif mainlist1forp[a+3] == 'P' and mainlist1forp[a+30] == 'P':
# Critical elif
                if mainlist1forp[a+4] == 'P' and mainlist1forp[a+40] !=
'P':
                    p_lerim.append([a, a + 10])
                    pl__p_ind.remove(a)
                    pl__p_ind.remove(a + 10)
                elif mainlist1forp[a+4] != 'P' and mainlist1forp[a+40] ==
'P':
                    p_lerim.append([a, a + 1])
                    pl__p_ind.remove(a)
                    pl__p_ind.remove(a+1)
return p_lerim
```

Main purpose of this complex if - elif structure is that check one step right and one step down of the 'P'(index='a') that is the first element of *list of p indices*. If right index*[a+1]* is 'P'and down one *[a+10]* is not then there is just one possibility that is *[a,a+1]* and if just the index *[a+10]* is P then p1 one ship has indices *[a,a+10].* The point that I callef 'Critical Elif' is the possibility of both *[a+1]* and *[a+10]* is equal 'P'. That means now we have to check *[a+2]* and [*a+20*]. It goes like this until the eight index has checked.

Making every move by order:

Problem: Both moves input files belongs two player shoud be rad by order and round by round.

Solution: Using nested for loop plays the game according to correct order.

```
For i in player 1 inputs:

    Check the i if it is a valid move or not
    Then do the player 1s shot
    For j in player 2 inputs :

            Check the i if it is a valid move or not
            Then do the player 1s shot
            Show boards
            Check what ships are sunk
            Check is there any winner or game is draw
            Break
```

Break: is most important part of this solution because it provides to skip next move of player 1 after player 2 make its move

3 Programmer’s Catalogue

Writing Function:

```
def writing(messageforwriting):
    with open('Battleship.out', 'a') as g:
        g.write(messageforwriting)
        g.write('\n')
```

find_indices()

```
def find_indices(list_to_check, item_to_find):
    indices = []
    for idx, value in enumerate(list_to_check):
        if value == item_to_find: indices.append(idx)
    return indices
```

locating_b()

```
def locating_b(mainlist, pl__b_ind):
    pl__b_ind = [int(k) for k in pl__b_ind]
    a = pl__b_ind[0]
    b1, b2 = [], []

    if mainlist[a+1] == 'B' and mainlist[a+2] == 'B':
        if mainlist[a+3] == 'B':
            b1 = pl__b_ind[0:4]
            b2 = pl__b_ind[4:9]
    elif mainlist[a+10] == 'B' and mainlist[a+20]:
        if mainlist[a+30] == 'B':
            b1 = [a, a+10, a+20, a+30]
            pl__b_ind.remove(a)
            pl__b_ind.remove(a+10)
            pl__b_ind.remove(a+20)
            pl__b_ind.remove(a+30)
            b2 = pl__b_ind
    return b1, b2
```

checking_errors()

This function checks the moves if they are invalid then prints a message if there is nor error it return True. So that move can be made. Otherwise it passes the next move.

```
def checking_error(move):  # This part for checking moves errors.
    columnnum = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
    try:
        if move.count(',') < 1: raise IndexError
        args = move.split(',')
        args1, args2 = args[0], args[1]
        if len(args1) == 0 or len(args2) == 0: raise IndexError
        args1 = int(args1)
        try: int(args2)
        except ValueError: pass
        else: raise ValueError
        assert args1 in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
        assert args2 in columnnum
    except IndexError:
        print('IndexError: Your input \'{}\' is invalid. Some arguments are
missing.\n'.format(move))
        writing('IndexError: Your input \'' + move + '\' is invalid. Some
arguments are missing.\n')
        pass
    except ValueError:
        print('ValueError: Your input \'{}\' is invalid.'.format(move),
end='')
        print('First argument must be numeric and, the second must be
uppercase letter from(A-J).\n')
        writing('ValueError: Your input \'' + move + '\' is invalid.First
argument must be numeric and,'
         ' the second must be uppercase letter from(A-J).\n')
        pass
    except AssertionError:
        print('AssertionError: Invalid Operation.\n')
        writing('AssertionError: Invalid Operation.\n')
    except Exception:
        print('kaBOOM: run for your life!\n')
        pass
    else: return True  # If there is no error it returns True.
```

hits()

When everythings is on its way it is time to make the shots this function check the given index in mainlist of opponent then if there is a piece of ships it make this value of index ’X’ other wise it will be ‘O’ .

```
def hits(dictt, index, mainlist, hiddenlist):  # I do my shots with this
function.
    if mainlist[index] == '-' or mainlist[index] == 'O':
        mainlist[index], hiddenlist[index] = 'O', 'O'
    else:
        mainlist[index], hiddenlist[index] = 'X', 'X'
        for i in dictt.keys():
            try: dictt.get(i).remove(index)
            except: pass
```

sunkships()

Determine any ship is sunk it checks every pieces index is shot or not . If it is it changes the ’-‘ to ‘X’ in the boards

```
def sunkships(dictt, C, B, D, S, P):  # I show which ships are sunk.
    if dictt['C'] == [] : C = 'X'
    if dictt['S'] == [] : S = 'X'
    if dictt['D'] == [] : D = 'X'
    if len(dictt['B1']) == 0 and len(dictt['B2']) == 0: B = 'X X'
    elif bool(len(dictt['B1']) == 0) ^ bool(len(dictt['B2']) == 0): B = 'X
-'
    count=0
    for i in ['P1', 'P2', 'P3', 'P4']:
        if dictt[i] == []: count += 1
    if count ==1: P = 'X - - -'
    if count == 2: P = 'X X - -'
    if count == 3: P = 'X X X -'
    if count == 4: P = 'X X X X'
    return C, B, D, S, P
```

winner()

It is a function that checks the remaining ships at the end of the every round . So that it can realize if one of the player wins or game is a draw.

```
def winner(dictt1,dictt2):  # Determining winner.
    ships1, ships2, keys1, keys2 = [], [], list(dictt1.keys()),
list(dictt2.keys())
    for i in keys1: ships1.extend(dictt1[i])
    for j in keys2: ships2.extend(dictt2[j])
    if ships1 == [] and ships2 == []:
        print('It is a Draw!\n')
        writing('It is a Draw!\n')
    elif ships1 == [] and ships2 != []:
        print('Player2 Wins!\n')
        writing('Player2 Wins!\n')
    elif ships1 != [] and ships2 == []:
        print('Player1 Wins!\n')
        writing('Player1 Wins!\n')
```

4 User’s Catalogue

Restrictions about program:

First of all format of move written in the input files is fatally important. It should be like this:

7,A;5,E;10,G…

Otherwise program gives an error messeage and do not operates your move because of the wrong format.

Be sure that write enough moves in input files to terminate the game. Because if there are not enough moves , the game may not be resulted.

When running the program in the command prompt be careful about name of the txt files and order of them . Otherwise program will not start.

Also program file(Assignment4.py) and txt files must be in the same directory otherwise program gives IO Error.

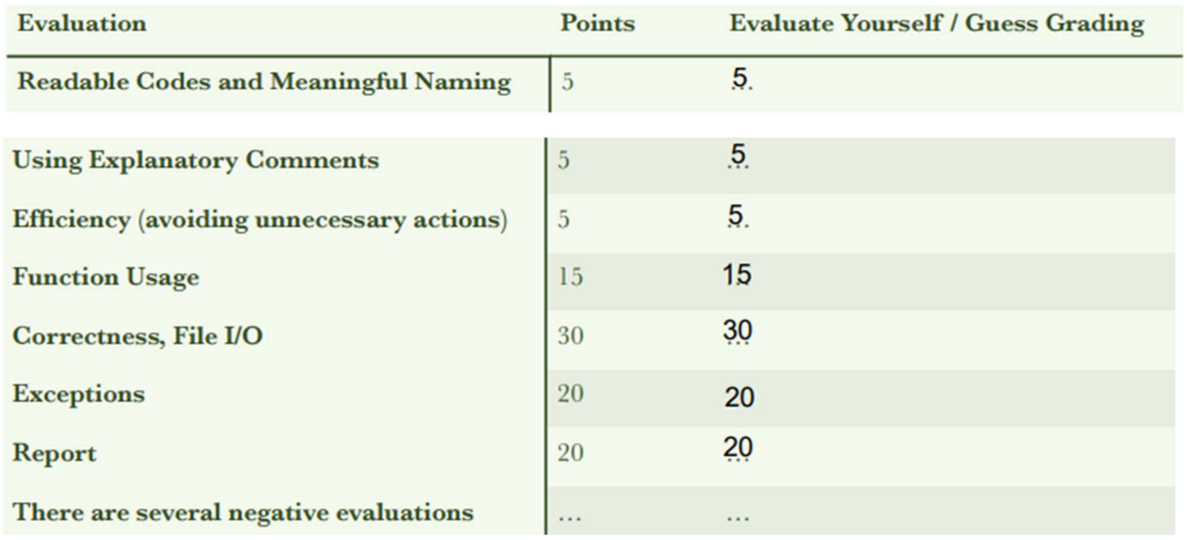
When you prepare this files :

1. Player1.txt
2. Player2.txt
3. Player1.in
4. Player2.in

All you have to do is writing command prompt this:

```
python3 Assignment4.py "Player1.txt" "Player2.txt" "Player1.in" "Player2.in"
```

The result will be provided in both command prompt and ‘Battleship.out’ which will be in the same directory with your files.



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