Assignment 4

Student Name: Ufuk Cefaker

ID: 2220356171

Lecture: BBM103

Delivery Date : 03/01/2023

Analysis :

In this assignment we have battleship game. 2 player are playing this game and that 2 player have their own board. In that board players are putting their ships on that board. Locations ,where ships are positoned , are selected by players . On this game there are 5 type of ships are positioned which their names are Carrier,Battleship,Destroyer,Submarine,Patrol Boat. Totaly 9 ships are positoned on the board their counts like: Carrier to 1, Battleship to 2, Destroyer to 1, Submarine to 1, Patrol Boat to 4. The purpose of the game is sinking all of opponents ships.

After the ships are positoned, player 1 is starting to game and choosing one row and column to shoot one of the positoned place. If in shooted place there is a part of any ship then it printed on table like “X” else , printed on the table like “O” after the shooting now it’s turn for player 2. Player 2 also shooted any positon like player 1’s table.

After the player 2’s shoot round is ending then another round is starting . Game keep going like this until game is finished. Game finished in 3 type of result. If one of the round player 1 can complete shooting all ships in player 2’s board then it’s turn for player 2 if player 2 also shoot all of the ship then the result is Draw . If just player 1 shooted all of them then player 1 Win will be printed . Also if player 2 shooted all of them then player 2 Win will be printed.

There could be some several mistakes in inputs ; such as IndexError,IOError,AssertionError,ValueError.

IndexError: Code gives ındexerror when there is a missing input. 4 input is needed for this assignment. When there is inputs less than 4 then this error occurs. Or one of the shooting coordinates are not defined it’s row or column.

IOError: Code gives this error when one of the file can not be opened due to one or more than one file not reachable.

AssertionError: Code gives this error when one of the shooting coordinates are not in row or column range.

ValueError: Code gives this error when one of the shooting coordinates row or column not in it’s type such as when there should be A to J , there is a number then ValueError occurs.

Design:

Code starts with import sys, because inputs are taking from cmd.

All codes indicates to try except block if any IOError or IndexError happens in the beginning of the code then except block is executed.

1-)

```
      def rearrange_txt_file_name(x): #this func is being used for deleting ""
s from input name
        x.replace("\"","")
    absent_sys_files = []  #this list is being used for how many input that
given by user
    try:
        player1_txt_file = sys.argv[1]
    except:
        absent_sys_files.append("Player1 ships locations")   #if this input
was not given by user then being added to list.
    try:
        player2_txt_file = sys.argv[2]
    except:
        absent_sys_files.append("Player2 ships locations")
    try:
        player1_in = sys.argv[3]
    except:
        absent_sys_files.append("Player1 shoots")
    try:
        player2_in = sys.argv[4]
    except:
        absent_sys_files.append("Player2 shoots")
    if len(absent_sys_files) != 0:  #if any command has not taken from user
then raise ındexerror.
        raise IndexError
    for x in(player1_txt_file,player2_txt_file,player1_in,player2_in): #in
here for every input; every \n will be deleted in every input list.
        rearrange_txt_file_name(x)  #for every input
```

For first part in the code, firstly taking input from user and all of them indicate to try except block for detecting which ındex has not given.

Absent_sys_files is going to be used for which ındex's are not given by using that if any ındex has not given IndexError is going to be raised and that list is going to be used.

And first function is going to be used for the " is going to be deleted in input names.

2-)

```
    wrong_input_list = []  #this list is being used for if there is no such
that file then being added to list that is wrong file.
```

```
    try:
        f = open(player1_txt_file,"r")  #reading input
        player1_ships_locations = f.readlines()
    except:
        wrong_input_list.append(player1_txt_file)
    try:
        f = open(player2_txt_file,"r") #reading input
        player2_ships_locations = f.readlines()
    except:
        wrong_input_list.append(player2_txt_file)
    try:
        f = open(player1_in,"r") #reading input
        player1_moves = f.readlines()
    except:
        wrong_input_list.append(player1_in)
    try:
        f = open(player2_in,"r") #reading input
        player2_moves = f.readlines()
    except:
        wrong_input_list.append(player2_in)

    if len(wrong_input_list) !=0:  #if any of them is wrong then raise
IOError.
        raise IOError

    f = open("OptionalPlayer1.txt","r")    #reading optional input
    OptionalPlayer1 = f.readlines()
    f = open("OptionalPlayer2.txt","r")   #reading optional input
    OptionalPlayer2 = f.readlines()
```

In second part, there is a list which is name wrong_input_list this is going to be used for if any input name can not openable in that directory then this name is going to be added that list. If any of them can not be openable (length of wrong_input_list is not equal to 0) then raise IOError.

If all of them openable then go to next part.

Open to OptionalPlayer1 and OptionalPlayer2 txt's. And OptionalPlayer1 and OptionalPlayer2 are going to be that txt's list which lines are included into that.

3-)

```
    for a in
(player1_ships_locations,player2_ships_locations,OptionalPlayer1,OptionalPlaye
r2): #this loop is being used for deleting \n from all inputs and turning to
list.
        for b in range(len(a)):
            a[b] = a[b].replace("\n","")
    def adding_empty_board(x): #this func is being used for creating empty
list for a row.
        for i in range(10):
```

```
            row = ["-","-","-","-","-","-","-","-","-","-"]
            x.append(row)

    player1_board , player2_board ,player1_hidden_board,player2_hidden_board=
[],[],[],[]  #for player boards 4 list is being created two of them for
printing two of them for controlling
    adding_empty_board(player1_board)  #player1_board will be ships locations
shower to programmer and end of the game this list is going to be used
    adding_empty_board(player2_board)  #player2_board will be ships locations
shower to programmer and end of the game this list is going to be used
    adding_empty_board(player1_hidden_board)  #player1_board will be ships
locations shower to user and before end of the game every round this list is
going to be used
    adding_empty_board(player2_hidden_board)   #player2_board will be ships
locations shower to user and before end of the game every round this list is
going to be used
```

For player1_ship_locations , player2_ship_locations, OptionalPlayer1 and OptionalPlayer2 :

For it's range(len of current variable):

Replacing \n by nothing.

A function detected for creating a list range 10x10 which function named a adding_empty_board. Row is a temporary variable which is being used for adding 10 element "-" to row. By looping totaly 10 list will be added into a list.And for every 10 list there is a 10 string "-" element is added.

player1_board, player2_board, player1_hidden_board and player2_hidden_board is used by that function

player1_board, player2_board are being used for secret list. Ships location are going to be hidden in that lists.

Other hidden board lists are going to be used for output table.

4-)

```
    def detecting_ships_locations(x,y):  #this func is being used for
detecting ships location and put them into their chechking board.
        for a in range(len(y)):
            temp_ship_locations = y[a].split(";")
            for b in range(len(temp_ship_locations)):
                if temp_ship_locations[b] == "":
                    pass
                else:
                    x[a][b] = temp_ship_locations[b]

    detecting_ships_locations(player1_board,player1_ships_locations) #to the
player1_board player1 's ships locations will be added to in the list.
    detecting_ships_locations(player2_board,player2_ships_locations)  #to the
player2_board player2 's ships locations will be added to in the list.
```

In this part detecting_ships_locations function is defined which is going to be used for detecting ships locations for a player. Two variable are taken by executing that function.

For a in range (length of second variable coming from executing that code):

temp_ship_locations = a'th element of y is splitted by ";"

for b in range (length of temp_ship_locations):

if b'th element of temp_ship_locations is equal to "":

pass

else:

b'th element of a and a'th element of x is setting to b'th element of temp_ship_locations

by using that function player1_board and player2_board are now detecting ships parts.

5-)

```
    players_ships_numbers_for1,players_ships_numbers_for2= dict(),dict()

    def detecting_location_by_optional_txt(player,players_ships_numbers): #in
this func; two ship type(battleship,patrol boat) have more than one ship thats
why we have locate every ships location:
        #thats why this function locate the every pointed ships.
        for a in range(len(player)): #for a in range len of the optional
player
            player[a]= player[a].split(";") #the optional variable row will be
splitted by ;
            if player[a][0][0] == "B": #if in this line starts with B
                #ship_type = "B"
                times_number = 4 # this variable being used for how many ships
will be choose for.
            else:
                #ship_type = "P"
                times_number = 2  # this variable being used for how many
ships will be choose for.
            ships_first_location = player[a][0].split(":") #this variable will
be used for first place of ships.

            if times_number == 2:   #if it is battleship:
                if player[a][1] == "right": #if it goes to the right:
                    players_ships_numbers[ships_first_location[0]] =
[[int(ships_first_location[1][:-2]), ord(ships_first_location[1][-1])-
64],[int(ships_first_location[1][:-2]), ord(ships_first_location[1][-1])-
63]]
                    #to this dict ships places are going to be added. In later
this dict is going to be used for controlling if the ship is sank or not.This
dict is going to be executed for 4 statement in the under.
```

```
                else:  #if it goes to down
                    players_ships_numbers[ships_first_location[0]] =
[[int(ships_first_location[1][:-2])+x, ord(ships_first_location[1][-1])-64]
for x in range(2)]
            else:   #if it is patrol boat:
                if player[a][1] == "right":  #if it goes to the right:
                    players_ships_numbers[ships_first_location[0]] =
[[int(ships_first_location[1][:-2]), ord(ships_first_location[1][-1])-64 + x]
for x in range(4) ]
                else:   #if it goes to down
                    players_ships_numbers[ships_first_location[0]] =
[[int(ships_first_location[1][:-2])+x, ord(ships_first_location[1][-1])-64]
for x in range(4)]

    for_optional_detect = "1"
    detecting_location_by_optional_txt(OptionalPlayer1,players_ships_numbers_f
or1) #battleship and patrol boat ships locations will be added
players_ships_numbers_for1 dict.
    for_optional_detect = "2"
    detecting_location_by_optional_txt(OptionalPlayer2,players_ships_numbers_f
or2) #battleship and patrol boat ships locations will be added
players_ships_numbers_for2 dict.
```

players_ships_numbers_for1,players_ships_numbers_for2 are being setted to empt dict.

detecting_location_by_optional_txt is a function that is going to be used for detecting ships parts locations specificly. By getting information from that part of the under the table which ship are sinked is being used.

2 variable are taken by using that function. First one is for Optional list, second one is for empty dictionary.

For a in range(length of player):

a'th element of player is setting to a'th element of player is splitted by ";"

if 0'th element of 0'th element of a'th element of player is equal to "B":

times_number is setting to 4

else:

times_number is setting to 2

ships_first_location is setting to 0'th element of a'th element of player is splitted by ":"

if times_number is equal to 2:

if 1'st element of a'th element of player is equal to "right":

players_ships_numbers[ships_first_location[0]] = [[int(ships_first_location[1][:-2]), ord(ships_first_location[1][-1])-64],[int(ships_first_location[1][:-2]), ord(ships_first_location[1][-1])-63]]

(as a result a ships parts is a key of that dict and its locations its value)

Else:

players_ships_numbers[ships_first_location[0]] = [[int(ships_first_location[1][:-2])+x, ord(ships_first_location[1][-1])-64] for x in range(2)]

(in here also happens same thing but detecting locations into down)

Else (times_number equal to 4):

If 1'st element of a'th element of player is equal to "right":

players_ships_numbers[ships_first_location[0]] = [[int(ships_first_location[1][:-2]), ord(ships_first_location[1][-1])-64 + x] for x in range(4)]

(happents same thing but loop is going for 4)

Else:

players_ships_numbers[ships_first_location[0]] = [[int(ships_first_location[1][:-2])+x, ord(ships_first_location[1][-1])-64] for x in range(4)]

(also happens same thing but going to down.)

for_optional_detect is setting to 1. Because by using that detecting which player should we added its dict.

Func is being called for player1 optionals.

for_optional_detect is setting to 2.

Func is being called for player2 optionals.

6-)

```
    player1_moves,player2_moves =
player1_moves[0].split(";"),player2_moves[0].split(";") #player moves is going
to be added into their player_moves list.
    def error_detecter(player_move,player_move_calculator): #this func is
being used for if there is any error in shooting input.
        control_list = ["A","B","C","D","E","F","G","H","I","J"] #this 2 list
is being used for if shooting line is true or not.
        global output_message,check_for_check #output_message is a printed
message for every round , check_for_check is being used for if there is any
error then this variable is going to be True.
        check_for_check = False #firstly there is no error.
        try:
            if len(player_move)-1< player_move_calculator: #if one of the
players shoot move is ended then not check the shoot.
                quit()
            if len(player_move[player_move_calculator]) ==0:  #if shoot move
is empty:
```

```
                raise IndexError("IndexError: row and colum coordinate could
not be defined because of nothing has typed to command.Wrong command is:
"+player_move[player_move_calculator])
            if "," not in player_move[player_move_calculator]: #if shoot move
is not be parted by comma:
                raise IndexError("IndexError: row and column coordinate could
not be defined,Please type a comma when you define a command.Wrong command is:
"+player_move[player_move_calculator])
            else:
                player_movements_location =
player_move[player_move_calculator].split(",") #shoot move splitted by comma.
            if len(player_movements_location) >2:  #if there is more than one
comma in shoot move :
                raise ValueError("ValueError: there is more than two
coordinate,Please type a command which has 2 dimension.Wrong command is:
"+player_move[player_move_calculator])
            if len(player_move[player_move_calculator]) <3: #if len of shoot
move is smaller than 3 which is wrong:
                if len(player_move[player_move_calculator]) == 1 or
len(player_move[player_move_calculator])==0: #if shoot move len is 0 or 1:
                    raise IndexError("IndexError: row and column coordinate
could not be defined,Please type defined column.Wrong command is:
"+player_move[player_move_calculator])
                if player_move[player_move_calculator][0] == "," and
player_move[player_move_calculator][0] ==
player_move[player_move_calculator][1]: #if shoot move is ,,
                    raise IndexError("IndexError: row and column coordinate
could not be defined,Please type defined column.Wrong command is:
"+player_move[player_move_calculator])
                if player_move[player_move_calculator][0] == ",":  #if shoot
move is ,... .. could be everything.
                    raise IndexError("IndexError: column coordinate could not
be defined because row is not typed in (A to J),Please type defined
column.Wrong command is: "+player_move[player_move_calculator])
                else: #if -, - will be everything , but column not typed:
                    raise IndexError("IndexError: row coordinate could not be
defined because row is not typed in .(1 to 10) just empty is typed,Please type
defined column.Wrong command is: "+player_move[player_move_calculator])
            if len(player_move[player_move_calculator]) >4: #if len of shoot
move is larger than 4:
                if len(player_movements_location[0]) > 2 and
len(player_movements_location[1]) >1: #if both of sides are not in their
boundaries:
                    raise AssertionError("AssertionError: row and column
coordinates could not be defined because of coordinates are not in
boundary,Please type defined valid command.Wrong command is:
"+player_move[player_move_calculator])
                elif len(player_movements_location[0]) > 2: #if just row
coordinate not in its boundary:
```

```
                    raise AssertionError("AssertionError: row coordinate could
not be defined because of coordinate is not in boundary,Please type defined
valid command.Wrong command is: "+player_move[player_move_calculator])
                elif len(player_movements_location[1]) > 1: #if just column
coordinate not in its boundary:
                    raise AssertionError("AssertionError: column coordinate
could not be defined because of coordinate is not in boundary,Please type
defined valid command.Wrong command is: "+player_move[player_move_calculator])
            if len(player_move[player_move_calculator]) ==3: #if shoot move
len is 3:
                if ord(player_movements_location[1]) >74: #ord is being used
in here; which is turning to ascii number of this character.
                    #in ascii 74 is "J" thats why if it is larger than 74
column not is in boundary:
                    raise AssertionError("AssertionError: columm coordinate
could not be defined because column is not typed in (A to J).Wrong output is:
"+player_move[player_move_calculator])
                if player_movements_location[0] in control_list: #if row
coordinate in A to J:
                    if player_movements_location[1] not in control_list: #if
column coordinate not in A to J:
                        raise ValueError("ValueError: row and column
coordinate could not be defined because row (1 to 10) and column (A to J) are
not typed in their intervals.Wrong command is:
"+player_move[player_move_calculator])
                    else:  #just row is wrong then:
                        raise ValueError("ValueError:row coordinate could not
be defined because row coordinate is not in its interval(1 to 10).Wrong
command is: "+player_move[player_move_calculator])
                if player_movements_location[1] not in control_list: #if just
column not in its boundary:
                    raise ValueError("ValueError: column coordinate could not
be defined because column coordinate is not in its interval(A to J).Wrong
command is: "+player_move[player_move_calculator])
            if len(player_move[player_move_calculator]) ==4: #if len of shoot
move is 4:
                try:
                    if len(player_movements_location[1]) >1: #if column len is
larger than 1:
                        if player_movements_location[0] == "0": #if also row
coordinate is 0 then both is wrong.
                            raise ValueError("ValueError: row and column
coordinates could not be defined because row is under the boundary and column
is not typed in (A to J).Wrong command is:
"+player_move[player_move_calculator])
                        else: #else just one of them which column is wrong.
                            raise ValueError("ValueError: column coordinate
could not be defined because column is not typed in (A to J).Wrong command is:
"+player_move[player_move_calculator])
```

```
                    else:
                        if int(player_movements_location[0]) >10:  #if column
coordinate is larger than 10:
                            raise AssertionError("AssertionError: row
coordinate could not be defined because row is not typed in (1 to 10).Wrong
output is: "+player_move[player_move_calculator])
                        elif ord(player_movements_location[1]) >74: #if ascii
turning of column coordinate is larger than 74(Which is "J"):
                            raise AssertionError("AssertionError: columm
coordinate could not be defined because column is not typed in (A to J).Wrong
output is: "+player_move[player_move_calculator])
                except ValueError as e: #if ValueError happens than raise
another ValueEror to write its output.
                    raise ValueError(e)
        except IndexError as e:
            check_for_check = True #which mean error is occurred.
            x = "\n\n"+str(e)
            output_message = output_message+ x # #this message x is being
added to output message.
        except ValueError as e:
            check_for_check = True  #which mean error is occurred.
            x = "\n\n"+str(e)
            output_message = output_message+ x  # #this message x is being
added to output message.
        except AssertionError as e:
            check_for_check = True  #which mean error is occurred.
            x = "\n\n"+str(e)
            output_message = output_message+ x  # #this message x is being
added to output message.
        except:
            check_for_check = True  #which mean error is occurred.
            x = "\n\n"+"Kaboom: run for your life."
            output_message = output_message+ x  # #this message x is being
added to output message.
```

In this part a function is assigned to checking if any error has in any player shooting. Function is being called by 2 variable: First one is player_move which is any player move i, Second is player_move_calculator which is current move of player (if any error happened before it increased by 1)

control_list is setting to a list which contains a strings to every element A to J

global output_message and check_for_check is being called.

Check_for_Check is setting to False.

Making try except block for checking if any error has current shooting.

If length of player_move -1 is less than player_move_calculator:

Quit() (because player moves ended)

If length of player_move_calculator’th element of player_move is equal to 0:

Raise IndexError(row and column not defined.)

If “,” not in player_move_calculator’th element of player_move:

Raise IndexError(row and column again not defined.)

Else:

player_movements_location is setting to player_move_calculator’th element of player_move is splitted by “,”

if length of player_movements_locations is greater than 2:

Raise ValueError(More than 2 coordinates has given)

If lenght of player_move_calculator’th element of player_move is less than 3:

If length of player_move_calculator’th element of player_move is equal to 1 or 0:

Raise IndexError(row and column not defined)

If 0’th element of player_move_calculator’th element of player_move is equal to “,” and second element is equal to 1 (“,,”):

Raise IndexError(row and column not defined)

If 0’th element of player_move_calculator’th element of player_move is equal to “,”:

Raise IndexError(column not defined)

Else:

Raise IndexError(row not defined)

If length of player_move_calculator’th element of player_move is greater than 4:

If length of 0’th element of player_movements_locations is greater than 2 and length of 1’th element of player_move_calculator’th element of player_move is greater than 1:

Raise AssertionError (row and column not in its range)(120,YY)

Elif length of 0’th element of player_movements_locations is greater than 2:

Raise AssertionError (row not in its range)(120,B)

Elif length of 1’th element of player_movements_locations is greater than 1:

Raise AssertionError(Column not in its range)(10,BB)

If length of player_move_calculator’th element of player_move is equal to 3:

If ord(1’st element of player_movements_location) is greater than 74: (in ascii 74 is equal to “J” if that number greater than this for example “Z”)

Raise AssertionError(column not in range)(5,Z)

If 0’th element of player_movements_location in control_list:

If 1’st element of player_movements_location not in control_list:

Raise ValueError(row and column in its type)(J,5)

Else:

Raise ValueError(row not its type)(J,J)

If 1’st element of player_movements_location not in control_list:

Raise ValueError(column not its type)(5,5)

If length of player_move_calculator’th element of player_move is equal to 4:

Try:

İf length of 1’st element of player_movements_location is greater than 1:

If 0’th element of player_movements_location is equal to “0”:

Raise ValueError(row and column not defined)(0,JJ)

Else:

If int of 0’th element of player_movements_location is greater than 10:

Raise AssertionError(row not in range)(11,J)

Elif ord of 1’st element of player_movements_location is greater than 74(“J” equal to 74):

Raise AssertionError(row and column not in range)(11,Z)

Except ValueError as e:

Raise ValueError(e)

Except errors:

For every message is included to output_message

7-)

```
    players_board,players_hidden_board = "Player1’s Board\t\t\t\t\tPlayer2’s
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"    ,   "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"
    #this strings in the upper will be used for while printing output these
strings will be added to top of table.Just players_board will be used in end
of raunds.
    #this 2 string in upper; are going to be used for writing the output of
the every round top.
```

```
    def output_for_table(for_player1_board,for_player2_board): #this func will
be used for arranging output message for entrance lists.
        def for_optional_case_ships(players_ships_numbers,player_board): #this
func will be used for ships that detected with optional txt's. This func
returns nothing but it detects if that ship sink or not.
            global battleship_controller,patrol_boat_controller #this
variables being used for if ship sink or not.
            battleship_controller = 0;patrol_boat_controller = 0 #for starting
these variables are 0 if ship sink then this variable will stay 0 again.
            for a in players_ships_numbers: #for optional cases dict will be
in loop.
                if a == "B1" or a == "B2": #if ship is battleship which mean.
                    shooting_calculator = 0 #this detects for how many part of
the ship sink.
                    for b in players_ships_numbers[a]:  #for that key's
variables:
                        if player_board[b[0]-1][b[1]-1] != "X":  #if it is not
shooted
                            shooting_calculator +=1 #add 1
                    if shooting_calculator ==0:  #if ship sink:
                        battleship_controller +=1 #then one of ships sink
                else:  #if ship is patrol boat:
                    shooting_calculator = 0 #this detects for how many part of
the ship sink.
                    for b in players_ships_numbers[a]:   #for that key's
variables:
                        if player_board[b[0]-1][b[1]-1] != "X":   #if it is
not shooted
                            shooting_calculator +=1  #add 1
                    if shooting_calculator ==0:  #if ship sink:
                        patrol_boat_controller +=1 #then one of ships sink

        global output_message #general output message is being called.
        if game_finished_detecter: #if game is finished which is will be
detected in mainloop.
            output_message = players_board +"" #if game is finished then
players open board will be added to output
        else:
            output_message = players_hidden_board + ""  #if game is not
finished then players hidden board will be added to output
        carrier1_counter,carrier2_counter
,destroyer1_counter,destroyer2_counter,submarine1_counter,submarine2_counter =
0,0,0,0,0,0 #this for counting ships left.
        for a in range(10): #because of grid size of row is10 that why loop
ended at 10
            if a ==9: #if a is equal to end of the loop
                output_message = output_message + str(a+1) #adding to row
number beginning of the table. if it is 10, space is not adding to table
            else:
```

```
                output_message = output_message + str(a+1)+" "
            for b in for_player1_board[a]: #firstly player1 board is base case
in loop (it could be hidden or player_board which will be choosen while func
is being called.)
                output_message = output_message + b + " " #If part of the ship
shooted or not, it does not matter.Just on the lists element will be added.
            if a ==9:  #if a is equal to end of the loop
                output_message = output_message + "\t\t"+str(a+1) #adding to
row number beginning of the table. if it is 10, space is not adding to table
            else:
                output_message = output_message + "\t\t"+str(a+1)+" "
            for b in for_player2_board[a]: #secondly player2 board is base
case in loop (it could be hidden or player_board which will be choosen while
func is being called.)
                output_message = output_message + b + " " #If part of the ship
shooted or not, it does not matter.Just on the lists element will be added.
            output_message = output_message + "\n" #end of the row new line is
being added.
            carrier1_counter,carrier2_counter = carrier1_counter+
player1_board[a].count("C"),carrier2_counter+ player2_board[a].count("C")
#this count processes will be showed that how many part of the that ship has
left.
            destroyer1_counter,destroyer2_counter = destroyer1_counter+
player1_board[a].count("D"),destroyer2_counter+ player2_board[a].count("D")
            submarine1_counter,submarine2_counter = submarine1_counter+
player1_board[a].count("S"), submarine2_counter+ player2_board[a].count("S")

        def transletor(x,y): #this will be return for in circumstances if
player shoot that ship or not.This returns for player1 board and player2
board.
            if x ==0 and y ==0: #if two player shoots their opponents ships.
                return ["X","X"]
            elif x ==0 : #if player1's ship is shooted.
                return["X","-"]
            elif y ==0: #if player2's ship is shooted.
                return ["-","X"]
            else: #if any ship is not shooted.
                return["-","-"]

        def battleship_controllers(battleship_controller): #this func will
return for just one player to showing how many battleship sink.
            if battleship_controller ==0:
                return ["-","-"]
            elif battleship_controller == 1:
                return ["X","-"]
            else:
                return ["X","X"]
```

```
        def patrol_boat_controllers(patrol_boat_controller): #this func will
return for just one player to showing how many patrol boat sink.
            if patrol_boat_controller == 0:
                return ["-","-","-","-"]
            elif patrol_boat_controller ==1:
                return ["X","-","-","-"]
            elif patrol_boat_controller ==2 :
                return ["X","X","-","-"]
            elif patrol_boat_controller ==3:
                return ["X","X","X","-"]
            else:
                return ["X","X","X","X"]

        output_message = output_message + "\n"  #new line is being added.
        ship_list = transletor(carrier1_counter,carrier2_counter) #transletor
func is being called for carrier ship number.
        output_message = output_message + "Carrier\t\t"+ship_list[0]
+"\t\t\t\tCarrier\t\t"+ship_list[1]+"\n" #under the table carrier shower is
being added by ship_list.

        for_optional_case_ships(players_ships_numbers_for1,player1_board)
#for_optional_case_ships is being called for player1 to learn if ship is sink
or not.
        battleship_control_list1 =
battleship_controllers(battleship_controller) #by battleship_controllers its
list will be returned.
        output_message = output_message +
"Battleship\t"+battleship_control_list1[0]+"
"+battleship_control_list1[1]+"\t\t\t\t" #list is being added to table.

        for_optional_case_ships(players_ships_numbers_for2,player2_board) #now
for player2 optional cases ship situations are being learned by this func.
        battleship_control_list2 =
battleship_controllers(battleship_controller)
        output_message = output_message +
"Battleship\t"+battleship_control_list2[0]+" "+battleship_control_list2[1] +
"\n"

        ship_list = transletor(destroyer1_counter,destroyer2_counter)
#transletor func is being called for Destroyer ship number.
        output_message = output_message + "Destroyer\t"+ship_list[0]
+"\t\t\t\tDestroyer\t"+ship_list[1]+"\n"

        ship_list = transletor(submarine1_counter,submarine2_counter)
#transletor func is being called for Submarine ship number.
        output_message = output_message + "Submarine\t"+ship_list[0]
+"\t\t\t\tSubmarine\t"+ship_list[1]+"\n"
```

```
        for_optional_case_ships(players_ships_numbers_for1,player1_board)
#for_optional_case_ships is being called for player1 to learn if ship is sink
or not.
        patrol_boat_control_list1 =
patrol_boat_controllers(patrol_boat_controller)
        output_message = output_message + "Patrol
Boat\t"+patrol_boat_control_list1[0]+" "+patrol_boat_control_list1[1]+"
"+patrol_boat_control_list1[2]+" "+patrol_boat_control_list1[3]+"\t\t\t"

        for_optional_case_ships(players_ships_numbers_for2,player2_board) #now
for player2 optional cases ship situations are being learned by this func.
        patrol_boat_control_list2 =
patrol_boat_controllers(patrol_boat_controller)
        output_message = output_message + "Patrol
Boat\t"+patrol_boat_control_list2[0]+" "+patrol_boat_control_list2[1]+"
"+patrol_boat_control_list2[2]+" "+patrol_boat_control_list2[3]
```

In this part firstly players_board and players_hidden board is setting to a string to above's of the tables.

A function is assigned which name is output_for_table . This function is going to be used for detecting currently output message to be able added to general_output_message. By 2 variable being called to this function which they are for_player1_board , for_player2_board.

Also another function is assigned in that function which name is for_optional_case_ships. This func is being used for for optional ships if they are sink or not.

battleship_controller and patrol_boat_controller is a int variables and this values showing that how many ship of them are sink.

Global output message is being called.

If game_finished_detector is True:

Players_board being added to output_message

Else:

players_hidden_board being added to output_message.

carrier1_counter,carrier2_counter
,destroyer1_counter,destroyer2_counter,submarine1_counter,submarine2_counter

are setting to 0

for a in range of 10:

values are being added to table and designed fitted.

Carrier1 , Carrier2,Destroyer1,Destroyer2,Submarine1,Submarine2 numbers are being added to their counter variables.

3 function are assigned for returning a list how many has left own their type.

Def transletor(x,y):

This func is being used for all ships but 2 optional ships. x,y values are ships counters.By using them returning a list how many ship left and by that list it printed into output table.

Def battleship_controllers(battleship_controller):

For this time player1 and player2 are different because of that this func being called 2 times.

battleship_controller is a int variable which is showing that how many ship sink.

Def patrol_boat_controller(patrol_boat_controller):

Again this func is being called 2 times. patrol_boat_controller is a int variable which is showing that how many ship sink.This difference is , this ship type has 4 ship that's why func is different than battleship.

Every ship number showers being added to output message by functions are being called. For optional ships functions are being called 2 times.

8-)

```
    game_finished_detecter = False #this variable will be used for if game is
finished or not.
    def shooting(target,player_board,player_hidden_board): #this function will
be used for shooting the target by player.
        target_list = target.split(",") #target variable splitted by comma.

        if player_board[int(target_list[0])-1][ord(target_list[1])-65] == "-":
#if there is no target in that positon:
            player_board[int(target_list[0])-1][ord(target_list[1])-65] = "O"
#change both hidden and normal board.
            player_hidden_board[int(target_list[0])-1][ord(target_list[1])-65]
= "O"
        else: #if one of ships in here.
            player_board[int(target_list[0])-1][ord(target_list[1])-65] = "X"
#target is shooted.
            player_hidden_board[int(target_list[0])-1][ord(target_list[1])-65]
= "X"
```

In this part firstly game_finished_detecter is setting to False because game is not finished.

A function is assigned which is name is shooting. By this function calling also 3 variable coming by this function which they are target,player_board,player_hidden_board.

target_list is setting to target is splitted by ","

If on that current location there is no any part of the any ship :

player_board and player_hidden board's on that current location changes to "O"

else:

player_board and player_hidden board's on that current location changes to "X"

9-)

```
    output_for_table(player1_board,player2_board)
    output_for_table(player1_hidden_board,player2_hidden_board)
    general_output_message = "Battle of Ships Game\n\n"  #this variable is for
output message. This variable will be our output.
    move1_calculator ,move2_calculator = -1, -1 #this move calculator's start
by -1 because in every loop's starting it increased by 1 .
```

for hidden and players board output_for_table is being called.

general_output_message is going to be final output message that's why current output messages are going to be added to that string.

Move1_calculator and move2_calculator are setting to -1 because in every turn this variables are being increased by 1.

10-)

```
    for mainloop in range(max(len(player1_moves),len(player2_moves))): #this
is general loop looping by max of one of the players moves:
        general_output_message = general_output_message + "Player1's
Move\n\nRound : "+str(mainloop+1)+"\t\t\t\t\tGrid Size:
10x10\n\n"+output_message
        #player1's starts every round thats why above strings added to general
output message.
        tester = True #this variable being used for if there is any mistake in
player_in shoot list then increasing the player moves.
        output_message = "" #output message is being used for adding every
message to general output message.
        while tester:
            tester = False
            move1_calculator +=1
            error_detecter(player1_moves,move1_calculator)
            if check_for_check == True: #if there is any error then:
                tester = True
        general_output_message = general_output_message +
output_message+"\n"+"\nEnter your move: "+str(player1_moves[move1_calculator])
+"\n"
        if move1_calculator >= len(player1_moves)-1: #if player1 moves ended
before game ended:
            pass
        else: #else shoot the target:
            shooting(player1_moves[move1_calculator],player2_board,player2_hid
den_board)
```

In this part main loop is starting. Loop is going for max of one of the player moves.

For every turn player1's tables above is being added to general_output_message (round number etc.).

tester setting to True

output message is setting to empyty string.

While tester is True:

tester is setting to False

move1_calculator is increased by 1

error detecter is being called by player1_moves and move1_calculator.

If check_for_check is True:

tester is setting to True

(Until correct shooting comes to player move this loop returns)

If move1_calculator is greater than length of player1_moves -1:

Pass

Else:

shooting function is being called by current player1_moves ,player2_board,player2_hidden_board.

(player2 boards being called because that board is being shooted.)

11-)

```
        general_output_message = general_output_message + "\nPlayer2's
Move\n\nRound : "+str(mainloop+1)+"\t\t\t\t\tGrid Size: 10x10\n\n"
        output_for_table(player1_board,player2_board)  #second time output
table is calling because it changes after shooting.
        output_for_table(player1_hidden_board,player2_hidden_board)
        general_output_message = general_output_message + output_message

        tester = True
        output_message = ""
        while tester:
            tester = False
            move2_calculator +=1
            error_detecter(player2_moves,move2_calculator)
            if check_for_check == True:
                tester = True
        general_output_message = general_output_message + output_message +
"\n"
        output_for_table(player1_board,player2_board) #third is being called
because if there is any error then index's line is changing.
        output_for_table(player1_hidden_board,player2_hidden_board)
        general_output_message = general_output_message +"\nEnter your move:
"+str(player2_moves[move2_calculator])+"\n\n"
        if move2_calculator >= len(player2_moves)-1:
            pass
        else:
            shooting(player2_moves[move2_calculator],player1_board,player1_hid
den_board)
```

In this part firstly player2's move and round number etc. are being added to general_output_message.

Then after the shooting again output_for_table function is being called for hidden and normal boards.

And output message being added to general_output_message.

tester is setting to True.

Setting output_message to empty string.

While tester is True:

Setting tester to False.

Move2_calculator is being increased by 1

error_detecter function is being called by player2_moves and move2_calculator .

if check_for_check is True:

setting tester to True.

(Until proper shooting finding this loop is turning.)

Output_message is being added to general_output_message.(If any error occurs then output message has its string error type.)

output_for_table is being called for normal and hidden boards again to detect which positons are being shooted.

"Enter your move: " string and its returning number is being added to general_output_message.

If move2_calculator is greater or equal to length of player2_moves -1:

Pass

Else:

shooting function is being called by current player move, player1_board and player1_hidden_board.

(because player1 board is being shooted.)

12-)

```
        def Game_finished_check(player_board): #this function is controlling
if one of the player shooted the every ship to his enemy.
            ship_counter = 0
            for a in player_board:
                for b in a:
                    if b == "C" or b == "B" or b == "D" or b == "S" or b ==
"P" :
                        ship_counter +=1
```

```
            if ship_counter ==0: #if there is not any ships left then game is
finished:
                return True
            else:
                return False
        check_for_player1 = Game_finished_check(player1_board) #checking for
player1
        check_for_player2 = Game_finished_check(player2_board) #checking for
player2
        if check_for_player1 and check_for_player2: #if both player shooted
them in same round then it is draw:
            game_finished_detecter = True
            output_for_table(player1_board,player2_board)
            general_output_message = general_output_message + "Game
Draw!\n\nFinal Information\n\n"+output_message
            break
        elif check_for_player1: #if just player2 shooted all ships to
player1's table: player2 wins
            game_finished_detecter = True
            output_for_table(player1_board,player2_board)
            general_output_message = general_output_message + "Player2
Wins!\n\nFinal Information\n\n"+output_message
            break
        elif check_for_player2: #if just player1 shooted all ships to
player2's table: player1 wins
            game_finished_detecter = True
            output_for_table(player1_board,player2_board)
            general_output_message = general_output_message + "Player1
Wins!\n\nFinal Information\n\n"+output_message
            break
```

This part is checking if game is finished or not.

A Function assigned which name is Game_finished_check(player_board)

This function returns True if a player's board's ships all shooted.

And if returns True loop is ending.

check_for_player1 and check_for_player2 is a variable that comes from Game_finished_check(by their board)

If both check_for_player1 and check_for_player2 are True:

game_finished_detector is setting to True.

Output_for_table is being called for normal boards.

"Game Draw " string is being added to general_output_message and output_message too.

Then breaking from loop.

Elif check_for_player1 is True:

game_finished_detector is setting to True.

Output_for_table is being called for normal boards.

“Player2 Wins! “ string is being added to general_output_message and output_message too.

(If player1_board’s all ships are sink which mean player2 wins that game.)

Elif check_for_player2 is True:

game_finished_detector is setting to True.

Output_for_table is being called for normal boards.

“Player1 Wins! “ string is being added to general_output_message and output_message too.

(If player2_board’s all ships are sink which mean player1 wins that game.)

13-)

```
    g = open("Battleship.out","w")
    g.write(general_output_message) #general message is being writed to in
this output file.
    g.close()
    #Sondaki errorları g.write yapmayı unutma  file io error olursa hata
ismini yakalamayı hallet.
    print(general_output_message) #also message is printed into the cmd.
```

In this part general_output_message is being printed into output file and command line too.

g is setting to writing command to Battleship.out file which is output file. And general_output_message is being printed to that file.

14-)

```
except IOError: #if there is any can not openable file:
    message = "IOError: "
    g = open("Battleship.out","w")
    if len(wrong_input_list) ==1: #if one of the file is not being opened:
        print(message+ wrong_input_list[0]+"is not reachable")
        g.write(message+ wrong_input_list[0]+"is not reachable")
    else:
        for x in wrong_input_list:
            message = message+ x+","
        print(message[:-1]+" are not reachable.")
        g.write(message[:-1]+" are not reachable.")
    g.close()
except IndexError: #if there is missing input:
    message = "IndexError: "
    g = open("Battleship.out","w")
    for x in absent_sys_files:
        message = message + x +" "
    print(message+"has not given as a input.")
```

```
    g.write(message+"has not given as a input.")
    g.close()
except: #except any mistakes:
    g = open("Battleship.out","w")
    g.write("Kaboom: run for your life.")
    g.close()
    print("Kaboom: run for your life.")
```

In this part ,which part is final part, any error occurs in the beginning of the code these except blocks are being executed.

Except IOError:

message is setting to “IOError: “

g is setting to writing command to output file.

if length of wrong_input_list is equal to 1: (If just one file can not be openable)

message + first element of wrong_input_list + “is not reachable” is printed into output file and command line too.

Else: (More than 1 file are not openable)

For x in wrong_input_list:

x and “,” are being added to message.

message but its last index + “ are not reachable “ is printed into output file and command line too.(the reason that last index not added is last element is “,” but we dont want this index.)

g is closing.

Except IndexError:

message is setting to “IndexError “

g is setting to writing command to output file.

for x in absent_sys_files: (absent input files)

x and “ “ are being added to message.

message + “has not given as a input” is being printed into output file and command line.

g is closing.

Except: (for any mistakes)

g is setting to writing command to output file.

“Kaboom: run for your life.” is printed into output file and command line too.

And g is closing.

Programmer's Catalogue:

I spent 15 20 hour to complete all lines of the code which was very important and most time taken project happened to me. Firstly i thought that i could gathered all tables into a string then i figured that out it was a big mistake why i was not gonna manage to shoot a location. That's why i turned into a list which has 10 element of the list and in every element there is 10 element of "-" . And by getting every location of the ships in that list elements are changed. By every turn if any part of the ship has shooted ,then this part is changed by "X".

Firstly all informations about ships locations , function has been made , after that main loop is started which loop turns by max length of one of the player moves.

Turn starts with Player1 and firstly its shooting target has been selected . This is selected by player1_move_calculator variable. Firstly it starts with 0 then if on that index there is a error to shoot that position then this variable increase it goes like this until it finds correct input.

When it finds correct input it shoots that positon then turn goes to player2

After shooting table is printed then same stages happens to player2 too.

Loop is keep going like that until one of the board there is no part of ship left. On this situation there is a possible 3 results:

Player 1 Win:

If player1 shooted all of them then turn goes to player2 to giving a chance to that player and if player2 could not shoot all of them then which mean Player1 wins the game.

Player2 Win:

It happens like that if just player 2 manage to finish shooting all of ships parts.

Draw:

If at the same round players both could manage sink all of the opponents ships.

All possible function calls showed at the design part by specifically.

Firstly output starts like this:

Battle of Ships Game

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

At the end of the game it ends like this:

Player2 Wins!

Final Information

Player1's Board Player2's Board

A B C D E F G H I J A B C D E F G H I J

1 O O O O - O X - O O

2 - O O O X O X O O O

3 - X - O X O X X X O

4 O X - O X O X O - O

5 O O O O X O X O - O

6 - X X X X - O O O O

7 O O O O O O X X X O O

8 O O - O O O O - O X

9 - - O - X X O O O X

10O X X O O - O - O X

1 - - O O O O O - O X

2 D O X X X X X O O X

3 D O O O O O O O O X

4 X O X X O O O O O O

5 - - O - O O X X B X

6 O O O O - - O O O -

7 O X O O P X O O O O

8 O B - O O O O X O O

9 - X O X X O O X O -

10O X O O O O O O O O

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

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

User Catalogue:

If you want to execute this program you can use any interpreter to execute such as cmd or inside terminal of the visual studio code. You have to be in the same directory where Assignment4.py is .You should use like this to execute that program:

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

4 input are needed to execute that program if you add less than 4 input then code will give IndexError.

You can change the winner of the game by changing the Player1.in or Player2.in file . Or you can change the locations of the ships where are positoned by changing Player1.txt or Player2.txt file.

You can add wrong inputs to Player1.in or Player2.in in one condition : keeping the turn shoot at the same number. Code would figure that out which coordinates are wrong and can not be shootable. After figuring that out it will also printed that it gives error and it goes to next input.

Grading Table:

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	...	0