Hacettepe University
Department of Computer Engineering

BBM103 Assignment 4 Report
Battle of Ships

İldeniz ÇELEBİ

b2210356013

03.01.2023

Concents

Analysis

Battleship is a two-player strategy game played on a grid. Each player has a fleet of ships that they place on their grid at the start of the game. The objective of the game is to sink all of the opponent's ships by correctly guessing the locations of the ships on the opponent's grid.

To begin a turn, a player announces a coordinate on the opponent's grid and their opponent announces whether a ship is present at that coordinate or not. If a ship is present, it is a hit. If not, it is a miss. The player continues their turn until they miss, and then the turn passes to the other player. The game ends when one player sinks all of the other player's ships.

There are different strategies that players can use to try to win the game. For example, a player might try to search for ships systematically, starting from one corner of the grid and working their way across and down until they have checked every coordinate. Alternatively, a player might try to use logic to deduce the locations of the ships based on the hits and misses they have obtained.

The main purpose of assignment 4 is to show the game that has been played by documenting the moves.

One of the parts that is the most important, and the main purpose of the assignment is debugging.

In this assignment, we are asked to design a code that extracts and solves specific errors for files and moves, and prints kaBOOM for errors that may occur other than these.

If an incorrect move is entered during the moves, what we have to do is print the error, then make the same player play his next move.

In this way, when all a player's ships are sunk, that player wins and the game is ended by printing the final information. If all the ships of both of them sank in the same round, the game ends in a draw.

Design

This is the beginning of the code. Its starts with “import sys” to use “sys.argv”. The file names entered as a command line argument were assigned to specific names and these names were used when opening and closing the file in the code. This try except block checks whether a sufficient number of command line arguments have been entered and prints the appropriate message without further action if they have not been entered.

```
import sys
try: #checking whether the correct number of command line arguments has been
entered
    player1txt=sys.argv[1]
    player2txt=sys.argv[2]
    player1in=sys.argv[3]
    player2in=sys.argv[4]
except:
    print("You entered less command line arguments than excepted.")
    f=open("Battleship.out","w")
    f.write("You entered less command line arguments than excepted.")
    f.close()
```

If there are no errors in the number of arguments, enter the else block and check whether the files you are trying to open are accessible. This is provided by the try else blocks, which are separate for each file. Thanks to the fact that it is separate, we can determine which file(s) are not accessible

```
else: #the try blocks in else check whether the files entered as command line
arguments have been opened if there is no problem with the amount of command
line arguments
    try:
        filenames="" #the string named filenames represents incorrectly
entered file names
        namecount=0 #the string named namecount represents the amount of files
entered incorrectly
        f= open(player1txt, "r", encoding="utf-8")
        f.close()
    except:
        namecount+=1
        filenames+=f"{player1txt}, "
    try:
        f= open(player2txt, "r", encoding="utf-8")
        f.close()
    except:
        namecount += 1
        filenames+=f"{player2txt}, "
    try:
        f=open("OptionalPlayer1.txt", "r", encoding="utf-8")
        f.close()
    except:
        namecount += 1
        filenames+="OptionalPlayer1.txt, "
    try:
        f=open("OptionalPlayer2.txt", "r", encoding="utf-8")
        f.close()
    except:
        namecount += 1
        filenames += "OptionalPlayer2.txt, "
    try:
        f = open(player1in, "r")
        f.close()
    except:
        namecount += 1
        filenames+=f"{player1in}, "
    try:
        f=open(player2in,"r")
        f.close()
    except:
        namecount += 1
        filenames += f"{player2in}, "
```

If there are files that are not accessible, they are customized as one or more than one, appropriate error messages are printed. “filenames[:-2]” allows to remove the trailing comma and space added to the string.

```
if namecount==1:
    print(f"IOError: input file {filenames[:-2]} is not reachable.")
    f=open("Battleship.out","w")
    f.write(f"IOError: input file {filenames[:-2]} is not reachable.")
    f.close()
elif namecount>1:
    print(f"IOError: inputfiles {filenames[:-2]} are not reachable.")
    f = open("Battleship.out", "w")
    f.write(f"IOError: inputfiles {filenames[:-2]} are not reachable.")
    f.close()
```

If all files were opened without problems, the “namecount” will be “0”. The entire part of the game and the code that does the actual work is located in this block.

```
elif namecount==0: #if there is no problem with the files, it enters this
block and the actual code works
```

Due to a possible error, i put all the remaining code in a try except block. If an error occurs other than the specified errors, the except block at the end of the code will catch this error and the message specified in that part will be printed.

```
try:
```

The necessary dictionaries to be used later.

“ship1dict” is a dictionary containing the coordinates of the ships of the player1 and which the player2 will use while playing the game.

“ship2dict” is a dictionary containing the coordinates of the ships of the player2 and which the player1 will use while playing the game.

An example of a shipdict in which the coordinates in it do not represent the truth coordinates will look like this: {1 : [-,-,-,-,C,-,-,-,-,-], 2 : [-,-,-,-,P,-,-,B,-,-], … , 10 : [-,-,-,-,-,-,-,-,-,-]}

“board1dict” and “board2dict” are created in the same logic as shipdict and have only hyphens in the list, where the hyphens will change as “X” and “O” according to the moves while the game is being played.

“board1dict” will be changed by player2's moves, “board2dict” will be changed by player1's moves.

```
boardsdict={}
ship1dict={}
ship2dict={}
board1dict={}
board2dict={}
```

The code from which the boarddicts are created.

```
for n in range(1,11):
    board1dict[n]=["-","-","-","-","-","-","-","-","-","-"] #this dictionary
is the dictionary where the hyphens will change to X and O according to the
moves of player2
for n in range(1,11):
    board2dict[n]=["-","-","-","-","-","-","-","-","-","-"] #this dictionary
is the dictionary where the hyphens will change to X and O according to the
moves of player1
```

The function by which the coordinates are transferred to the dictionary by reading the files. (After opening the files below, the function will be called with the correct dictionaries.)

```
def create_ship_dict(sözlük): #with this function, dictionaries containing
ship coordinates are created
    ship_coordinates = f.readlines()
    for i in range(len(ship_coordinates)):
        ship_coordinates[i] = ship_coordinates[i].replace("\n", "")
        ship_coordinates[i] = ship_coordinates[i].split(";")
        for x in ship_coordinates[i]:
            if x == "":
                index = ship_coordinates[i].index(x)
                ship_coordinates[i].remove(x)
                ship_coordinates[i].insert(index, "-")
    a = 0
    for rownum in range(1,11):
        sözlük[rownum] = ship_coordinates[a]
        a += 1
```

These two functions will come in handy when obtaining coordinates from dictionaries and the opposite.

```
def inttolet(integer): # function which takes integer as input and returns
letter
    res = {0 : "A",1 : "B",2 : "C",3 : "D",4 : "E",5 : "F",6 : "G",7 : "H",8 :
"I",9 : "J",10: "K",11: "L",12: "M",13: "N",14: "O",15: "P",16: "Q",17:
"R",18: "S",19: "T",20: "U",21: "V",22: "W",23: "X",24: "Y",25: "Z"}
    return res[integer]

def lettoint(letter): # function which takes letter as input and returns
integer
    res = {"A" : 0,"B" : 1,"C" : 2,"D" : 3,"E" : 4,"F" : 5,"G" : 6,"H" : 7,"I"
: 8,"J" : 9,"K" : 10,"L" : 11,"M" : 12,"N" : 13,"O" : 14,"P" : 15,"Q" : 16,"R"
: 17,"S" : 18,"T" : 19,"U" : 20,"V" : 21,"W" : 22,"X" : 23,"Y" : 24,"Z" : 25}
    return res[letter]
```

The main purpose of this function is to create a dictionary that will make it easier for us to group ships with more than one quantity and determine whether the ship has sunk. The coordinates of the ships are added to the lists in the dictionary defined in the code in a way that is appropriate.

```
def shipship(shipdict): #a function for creating a dictionary with ship
coordinates, the values of keys with ship names

ships={"Carrier":[],"Battleship":[],"Destroyer":[],"Submarine":[],"PatrolBoat"
:[]}
    for number in shipdict:
        for index in range(len(shipdict[number])):
            if shipdict[number][index]== "C":
                ships["Carrier"].append(str(number)+inttolet(index))
            if shipdict[number][index]== "B":
                ships["Battleship"].append(str(number)+inttolet(index))
            if shipdict[number][index]== "D":
                ships["Destroyer"].append(str(number)+inttolet(index))
            if shipdict[number][index]== "S":
                ships["Submarine"].append(str(number)+inttolet(index))
            if shipdict[number][index]== "P":
                ships["PatrolBoat"].append(str(number)+inttolet(index))
    return ships
```

Opening the "player1txt" and "player2txt" files and calling functions which are create_ship_dict and shipship to make or update dictionaries.

```
with open(player1txt, "r", encoding = "utf-8") as f:
    create_ship_dict(ship1dict) # dictionary containing the coordinates of the
player1's ships, player2 will use this dictionary
    ships1=shipship(ship1dict) # the dictionary of player 1, which contains a
```

```
ship's coordinates together in a list, player2 will use this dictionary
with open(player2txt, "r", encoding="utf-8") as f:
    create_ship_dict(ship2dict) # dictionary containing the coordinates of the
player2's ships, player1 will use this dictionary
    ships2 = shipship(ship2dict) # the dictionary of player 2, which contains
a ship's coordinates together in a list, player1 will use this dictionary
```

This function is one of the most important parts of the code.

It separates the coordinates of two separate Battleships with optional texts and converts the list, which is the values of the battleship, into a new list, with two lists which each of one represent a ship in it.

In the same logic for Patrol Boats, there are four lists representing four ships separately inside a list.

```
def Optional(dictionary): # the function for grouping ships with more than one
quantity separately
    Blist=[]
    B1list=[] # the list to which the coordinates of the first Battleship will
be added
    B2list=[] # for the second one
    Plist=[]
    P1list=[] # the list to which the coordinates of the first Patrol Boat
will be added
    P2list=[] # for the second one
    P3list=[] # for the third one
    P4list=[] # for the fourth one
    ship = f.readlines()
    for i in range(len(ship)):
        ship[i]=ship[i].replace("\n","")
        if ship[i][0]=="B":
            if ship[i][-2]=="t": #for right ---> t
                B1list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                B1list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 1))
                B1list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 2))
                B1list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 3))
                Blist.append(B1list)
            if ship[i][-2]=="n": #for right ---> n
                B2list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                B2list.append(str(int(ship[i].split(",")[0][3:])+1) +
(ship[i].split(",")[1][0]))
                B2list.append(str(int(ship[i].split(",")[0][3:])+2) +
(ship[i].split(",")[1][0]))
                B2list.append(str(int(ship[i].split(",")[0][3:])+3) +
(ship[i].split(",")[1][0]))
                Blist.append(B2list)
        if ship[i][0] == "P" and ship[i][1]=="1":
            if ship[i][-2] == "t": #for right ---> t
```

```
                P1list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P1list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 1))
                Plist.append(P1list)
            if ship[i][-2]=="n": #for right ---> n
                P1list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P1list.append(str(int(ship[i].split(",")[0][3:])+1) +
(ship[i].split(",")[1][0]))
                Plist.append(P1list)
        if ship[i][0] == "P" and ship[i][1] == "2":
            if ship[i][-2] == "t": #for right ---> t
                P2list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P2list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 1))
                Plist.append(P2list)
            if ship[i][-2] == "n": #for right ---> n
                P2list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P2list.append(str(int(ship[i].split(",")[0][3:]) + 1) +
(ship[i].split(",")[1][0]))
                Plist.append(P2list)
        if ship[i][0] == "P" and ship[i][1] == "3":
            if ship[i][-2] == "t": #for right ---> t
                P3list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P3list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 1))
                Plist.append(P3list)
            if ship[i][-2] == "n": #for right ---> n
                P3list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P3list.append(str(int(ship[i].split(",")[0][3:]) + 1) +
(ship[i].split(",")[1][0]))
                Plist.append(P3list)
        if ship[i][0] == "P" and ship[i][1] == "4":
            if ship[i][-2] == "t": #for right ---> t
                P4list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P4list.append(str(ship[i].split(",")[0][3:]) +
inttolet(lettoint(ship[i].split(",")[1][0]) + 1))
                Plist.append(P4list)
            if ship[i][-2] == "n": #for right ---> n
                P4list.append(str(ship[i].split(",")[0][3:]) +
(ship[i].split(",")[1][0]))
                P4list.append(str(int(ship[i].split(",")[0][3:]) + 1) +
(ship[i].split(",")[1][0]))
                Plist.append(P4list)
    dictionary["Battleship"]=Blist
    dictionary["PatrolBoat"]=Plist
    return dictionary
```

Since the optional texts will not be written as a command line argument, the file will be written as without using "sys. argv".

Assuming that the file is reachable i opened the files

Just below, I have called the function which uses these files.

```
with open("OptionalPlayer1.txt", "r", encoding = "utf-8") as f:
    Optional(ships1) # Player1's dictionary, player2 will use it
with open("OptionalPlayer2.txt", "r", encoding = "utf-8") as f:
    Optional(ships2) # Player2's dictionary, player1 will use it
```

The generated lists will be used in the output and to show whether the ships have sunk.

```
#lists containing sinking ship information
carrier_info1 =    ["Carrier     -"]
destroyer_info1 = ["Destroyer   -"]
submarine_info1 = ["Submarine   -"]
battleship_info1 =["Battleship  - -"]
patrolboat_info1 =["Patrol Boat - - - -"]

carrier_info2 =    ["Carrier     -"]
destroyer_info2 = ["Destroyer   -"]
submarine_info2 = ["Submarine   -"]
battleship_info2 =["Battleship  - -"]
patrolboat_info2 =["Patrol Boat - - - -"]
```

The round represents the number of moves in the game.

```
round=1
```

This function prints to the output file and terminal which player has the turn, the round, the hidden boards of both players, the information about whether the ship was hit and the move to be made, respectively.

```
def output(move): #the function that will print the moves and their
informations with hidden boards
    son=""
    global round
    if count[0]%2!=0:
        print("Player1's Move\n")
        file.write("Player1's Move\n\n")
        print("Round : ",round,"\t\t\t\t\t","Grid Size: 10x10","\n")
        file.write(f"Round : {round}\t\t\t\t\tGrid Size: 10x10\n\n")
    else:
        print("Player2's Move\n")
        file.write("Player2's Move\n\n")
        print("Round : ",round,"\t\t\t\t\t","Grid Size: 10x10","\n")
        file.write(f"Round : {round}\t\t\t\t\tGrid Size: 10x10\n\n")
        round += 1
    print("Player1's Hidden Board\t\tPlayer2's Hidden Board")
    file.write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n")
    print("  A B C D E F G H I J\t\t  A B C D E F G H I J")
    file.write("  A B C D E F G H I J\t\t  A B C D E F G H I J\n")
    for number in board1dict.keys():
        string=""
        if number != 10:
            sayı = str(number)+" "
        else:
            sayı = str(number)
        for b1 in board1dict[number]:
            string+=b1+" "
        string = sayı + string + "\t\t" + sayı
        for b2 in board2dict[number]:
            string+=b2+" "
        son+=string+"\n"
    print(son)
    file.write(son+"\n")

line=carrier_info1[0]+"\t\t\t\t"+carrier_info2[0]+"\n"+battleship_info1[0]+"\t
\t\t\t"+battleship_info2[0]+"\n"+destroyer_info1[0]+"\t\t\t\t"+destroyer_info2
[0]+"\n"+submarine_info1[0]+"\t\t\t\t"+submarine_info2[0]+"\n"+patrolboat_info
1[0]+"\t\t\t"+patrolboat_info2[0]
    print(line+"\n")
    file.write(line+"\n\n")
    print(f"Enter your move: {move}\n")
    file.write(f"Enter your move: {move}\n\n")
```

This function will be called only in the final, and the function that will print the last status of the tables and who won.

```
def final_output(winner): # the function that will be called last and print
who won by giving the final information
    son=""
    print(winner,"\n")
    file.write(winner+"\n\n")
    print("Final Information","\n")
    file.write("Final Information\n\n")
    print("Player1’s Board\t\t\t\tPlayer2’s Board")
    file.write("Player1’s Board\t\t\t\tPlayer2’s Board\n")
    print("  A B C D E F G H I J\t\t  A B C D E F G H I J")
    file.write("  A B C D E F G H I J\t\t  A B C D E F G H I J\n")
    for number in board1dict.keys():
        string=""
        if number != 10:
            sayı = str(number)+" "
        else:
            sayı = str(number)
        letter = -1
        for b1 in board1dict[number]:
            letter += 1
            if b1=="-":
                if ship1dict[number][letter]!="-":
                    b1=ship1dict[number][letter]
                    string += b1 + " "
                else:
                    string += b1 + " "
            else:
                string += b1 + " "
        string = sayı + string + "\t\t" + sayı
        letter2=-1
        for b2 in board2dict[number]:
            letter2 += 1
            if b2 == "-":
                if ship2dict[number][letter2] != "-":
                    b2 = ship2dict[number][letter2]
                    string += b2 + " "
                else:
                    string += b2 + " "
            else:
                string += b2 + " "
        son += string + "\n"
    print(son)
    file.write(son+"\n")

line=carrier_info1[0]+"\t\t\t\t"+carrier_info2[0]+"\n"+battleship_info1[0]+"\t\t\t\t"+battleship_info2[0]+"\n"+destroyer_info1[0]+"\t\t\t\t"+destroyer_info2[0]+"\n"+submarine_info1[0]+"\t\t\t\t"+submarine_info2[0]+"\n"+patrolboat_info1[0]+"\t\t\t\t"+patrolboat_info2[0]
    print(line)
    file.write(line+"\n")
```

The function takes in the following arguments:

move: a string representing the player's move, in the format "row,column" (e.g. "3,A")

shipdict: a dictionary representing the positions of the ships on the board

boarddict: a dictionary representing the current state of the board

ships: a dictionary containing information about the ships in the game

carrier_info: a list containing information about the carrier ship

destroyer_info: a list containing information about the destroyer ship

submarine_info: a list containing information about the submarine ship

battleship_info: a list containing information about the battleship

patrolboat_info: a list containing information about the patrol boat

In the function, the output function is called first and the current status is printed. Then the move is played and checked to see if there is a ship that has been hit with the for loop. at the end of the function, the ship information is checked to see if all the ships have been hit and the course of the game is determined.

```
def
time_to_play(move,shipdict,boarddict,ships,carrier_info,destroyer_info,submari
ne_info,battleship_info,patrolboat_info): # the function in which the game
will be played
    m=0
    output(move) # first, the current status will be printed, then the moves
will be played
    if shipdict[int(move.split(",")[0])][lettoint(move.split(",")[1])]=="-":
        boarddict[int(move.split(",")[0])][lettoint(move.split(",")[1])]="O"

    elif shipdict[int(move.split(",")[0])][lettoint(move.split(",")[1])]=="C":
        boarddict[int(move.split(",")[0])][lettoint(move.split(",")[1])] = "X"
        for s in ships["Carrier"]:
            if boarddict[int(s[:-1])][lettoint(s[-1])]=="X":
               m+=1
               if m==5:
                   carrier_info[0]= "Carrier     X"
            else:
                break

    elif shipdict[int(move.split(",")[0])][lettoint(move.split(",")[1])] ==
"B":
        boarddict[int(move.split(",")[0])][lettoint(move.split(",")[1])] = "X"
        for s in ships["Battleship"]:
            if (move.split(",")[0])+(move.split(",")[1]) in s:
                for s1 in s:
                    if boarddict[int(s1[:-1])][lettoint(s1[-1])] == "X":
                        m += 1
                        if m == 4:
                            if battleship_info[0][-3] == "-":
                                battleship_info[0] = "Battleship  X -"
                            else:
                                battleship_info[0] = "Battleship  X X"
                    else:
                        break

    elif shipdict[int(move.split(",")[0])][lettoint(move.split(",")[1])]=="D":
        boarddict[int(move.split(",")[0])][lettoint(move.split(",")[1])] = "X"
        for s in ships["Destroyer"]:
            if boarddict[int(s[:-1])][lettoint(s[-1])]=="X":
               m+=1
               if m==3:
                   destroyer_info[0]= "Destroyer   X"
            else:
                break

    elif shipdict[int(move.split(",")[0])][lettoint(move.split(",")[1])]=="S":
        boarddict[int(move.split(",")[0])][lettoint(move.split(",")[1])] = "X"
        for s in ships["Submarine"]:
            if boarddict[int(s[:-1])][lettoint(s[-1])]=="X":
               m+=1
               if m==3:
                   submarine_info[0]= "Submarine   X"
            else:
                break

    elif shipdict[int(move.split(",")[0])][lettoint(move.split(",")[1])]=="P":
        boarddict[int(move.split(",")[0])][lettoint(move.split(",")[1])] = "X"
        for s in ships["PatrolBoat"]:
            if (move.split(",")[0]) + (move.split(",")[1]) in s:
                for s1 in s:
                    if boarddict[int(s1[:-1])][lettoint(s1[-1])] == "X":
                        m += 1
                        if m == 2:
                            if patrolboat_info[0][-7] == "-":
                                patrolboat_info[0] = "Patrol Boat X - - -"
                            elif patrolboat_info[0][-5] == "-":
                                patrolboat_info[0] = "Patrol Boat X X - -"
                            elif patrolboat_info[0][-3] == "-":
                                patrolboat_info[0] = "Patrol Boat X X X -"
                            elif patrolboat_info[0][-1] == "-":
                                patrolboat_info[0] = "Patrol Boat X X X X"
                    else:
                        break
    if carrier_info1[0]=="Carrier     X" and battleship_info1[0]=="Battleship
X X" and destroyer_info1[0]=="Destroyer   X" and
submarine_info1[0]=="Submarine   X" and patrolboat_info1[0]=="Patrol Boat X X
X X":
        if count[0]%2!=0:
            pass
        else:
            if carrier_info2[0]=="Carrier     X" and
battleship_info2[0]=="Battleship  X X" and destroyer_info2[0]=="Destroyer   X"
and submarine_info2[0]=="Submarine   X" and patrolboat_info2[0]=="Patrol Boat
X X X X":
                winner="It is a Draw!"
                final_output(winner)
                raise AssertionError
            else:
                winner = "Player2 Wins!"
                final_output(winner)
                raise AssertionError
    elif carrier_info2[0]=="Carrier     X" and
battleship_info2[0]=="Battleship  X X" and destroyer_info2[0]=="Destroyer   X"
and submarine_info2[0]=="Submarine   X" and patrolboat_info2[0]=="Patrol Boat
X X X X":
        if count[0]%2!=0:
            pass
        else:
            if carrier_info1[0]=="Carrier     X" and
battleship_info1[0]=="Battleship  X X" and destroyer_info1[0]=="Destroyer   X"
and submarine_info1[0]=="Submarine   X" and patrolboat_info1[0]=="Patrol Boat
X X X X":
                winner = "It is a Draw!"
                final_output(winner)
                raise AssertionError
            else:
                winner = "Player1 Wins!"
                final_output(winner)
                raise AssertionError
```

A function that checks whether the move was entered incorrectly with the if blocks inside the try, and if it was entered correctly, enters into else and increases the count by one, calling the time_to_play function

```
def
error(move,shipdict,boarddict,ships,carrier_info,destroyer_info,submarine_info
,battleship_info,patrolboat_info):
    # the function that checks whether the moves have been entered incorrectly
    try:
        control=move.split(",")
        if len(control)<2:
            raise IndexError
        if control[0]=="":
            raise IndexError
        if control[1] == "":
            raise IndexError
        if len(control)>2:
            raise ValueError
        if type(control[0]) == "string" and type(control[1])=="integer":
            raise ValueError
        if type(control[0]) == "string" and type(control[1])=="string":
            raise ValueError
        if type(control[0]) == "integer" and type(control[1])=="integer":
            raise ValueError
        if int(control[0])>10:
            raise AssertionError
        if int(control[0])<1:
            raise AssertionError
        assert lettoint(control[1]) <= lettoint("J")
    except IndexError:
        print(f"IndexError: {move} is an unvalid move!")
        file.write(f"IndexError: {move} is an unvalid move!\n")
    except ValueError:
        print(f"ValueErrorIndexError: {move} is an unvalid move!")
        file.write(f"ValueErrorIndexError: {move} is an unvalid move!\n")
    except AssertionError:
        print(f"AssertionErrorIndexError: {move} is an unvalid move!")
        file.write(f"AssertionErrorIndexError: {move} is an unvalid move!\n")
    except:
        print("kaBOOM:run for your life!")
        file.write("kaBOOM:run for your life!\n")
    else:
        count[0]+=1 # the count increases by one for every move played
        # if the move is not incorrect, it calls the function to play the move

time_to_play(move,shipdict,boarddict,ships,carrier_info,destroyer_info,submari
ne_info,battleship_info,patrolboat_info)
```

```
def hamleler(playerin): # the function that reads the moves in the file and
turns them into a list
    f = open(playerin, "r")
    line = f.readlines()
    moves = ""
    for i in range(len(line)):
        line[i] = line[i].replace("\n", "")
        moves += line[i]
    moves = moves.split(";")
    moves.pop()
    f.close()
    return moves
x=len(hamleler(player1in)) # the number of moves of the player1
y=len(hamleler(player2in)) # the number of moves of the player2
```

This code appears to be the main game loop for the battleship game. The loop will continue as long as the number of moves played is less than or equal to the total number of. The code alternates between player 1 and player 2, with player 1 making a move if the count of moves played is even, and player 2 making a move if the count is odd.

```
with open("Battleship.out", "w", encoding = "utf-8") as file:
    print("Battle of Ships Game\n")
    file.write("Battle of Ships Game\n\n")
    a=0
    b=0
    count=[0] # represents the total number of moves played
    try:
        while count[0]<(x+y): # the loop continues as long as the number of
moves played is less than or equal to the total number of moves
            if count[0]%2==0: # if the count is an even number, the move turn
is player1's
                move1=hamleler(player1in)[a]
                # the function to be checked for errors is called first, if
there is no error, the function to play the game in it will be called

error(move1,ship2dict,board2dict,ships2,carrier_info2,destroyer_info2,submarin
e_info2,battleship_info2,patrolboat_info2)
                f.close()
                a+=1
            else: # if the count is an odd number, the move turn is player2's
                move2=hamleler(player2in)[b]
                # the function to be checked for errors is called first, if
there is no error, the function to play the game in it will be called

error(move2,ship1dict,board1dict,ships1,carrier_info1,destroyer_info1,submarin
e_info1,battleship_info1,patrolboat_info1)
                f.close()
                b+=1
    except AssertionError:
        pass
    except: #for the situation where the moves are over and no one wins
        print("The moves are over. No one won")
        file.write("The moves are over. No one won\n")
```

The except part of the try block at the beginning of the code.

```
except: # for the case of an unspecified error being
    print("kaBOOM: run for your life!")
    f=open("Battleship.out","w")
    f.write("kaBOOM: run for your life!")
    f.close()
```

Programmer's Catalogue

- For analyzing, I have spent approximately 7 hours. At first I had to read the pdf two or three times before I could do anything to understand.
- For designing, I have spent approximately 1day but it was constantly on my mind during the whole process and I was thinking about it all the time.
- For implementing, I have spent approximately 3 days.
- For testing, I have spent approximately 4-5 hours.
- For reporting, I have spent approximately 8 hours.
- I tried to make my code as clean and understandable as possible and added comment lines. In the design part, I also explained the goals by breaking the code into pieces.

User Catalogue

1) Create “Player1.txt”, “Player2.txt” files with player’s ship positions.
2) Create “Player1.in”, “Player2.in” files and write moves in them.
3) Execute the program by writing the terminal “python3 assignment4 Playerr1.txt Player2.txt Player1.in Player2.in”
4) You can see the documendation in “Battleship.out” file and also in terminal.