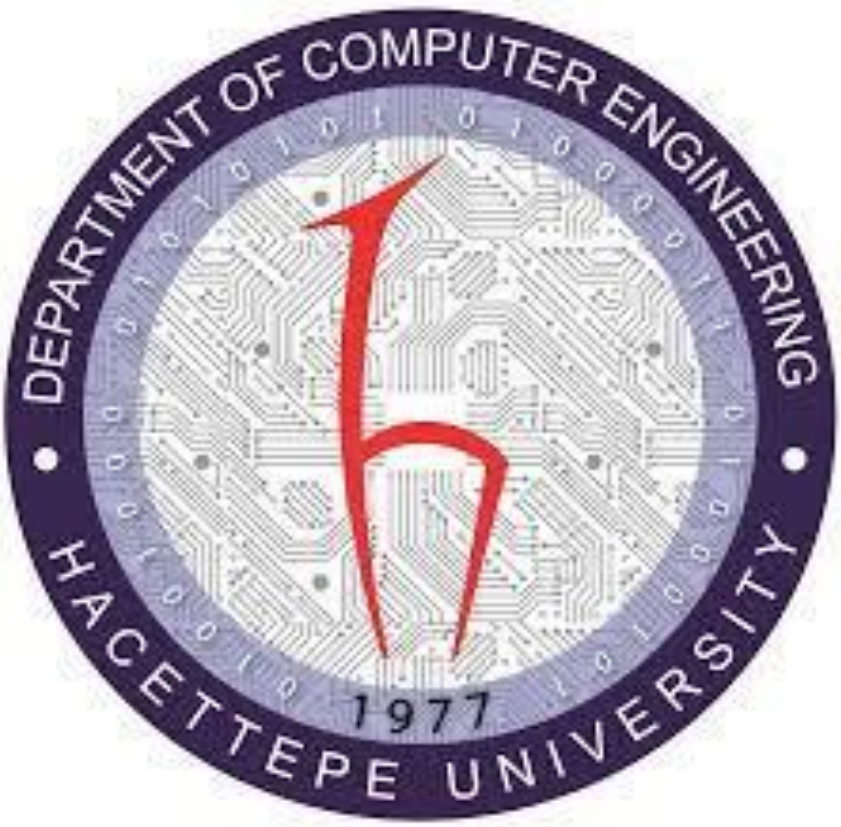
Hacettepe University

Department Of Computer Science

BBM103 Assignment 4 Report

Berke Abdullah Yıldız – 2210356100

01.01.2023



CONTENTS

Analysis

In this assignment, basically, the admiral sunk game will be played by taking the movements of the players from the files. First, a 10x10 table of players will be printed. It will then be stated that the ship was not hit with "-" signs. Each player's ship coordinates will be enclosed in a variable so they can be easily found later. The contents of the players' files will then be carefully read and their moves separated. The moves of the players will be made in turn. The lap after each move is displayed on the screen. Also, the number of moves in the game will be checked. All possible errors will be evaluated. For example, wrongly entered filenames, wrong moves, missing moves etc. At the end of the game, the player who explodes all the ships of the opposing side, except in the case of a draw, will win. In case of a draw, the other player will be given a chance and if the correct ship explodes, the game will end in a draw.

Design

1) **Imports, Reading Files and Exceptions**
   In this section, the files should be read, if there is a problem with the name of the files, an error should be given.

2) **Creation of Empty Player Tables and Moves**
   In this part, the tables of the players are created.

3) **Finding Ships**
   This section contains the locations of all the ships of player 1 and player 2.

4) **Ship Appearances**
   In this section, it is checked whether the ships have been hit according to the moves.

5) **Printing of Ships**
   In this section ships are printed both to the console and to the file.

6) **Making Tables**
   In this section, a table with numbers is created.

7) **Player's Moves**
   In this part, the moves of the players are performed and all necessary actions are taken.

8) **Final Situation**
   In this section, the state of the last ships and the state of the game are printed.

9) **Game's Move Commands and Exceptions**
   In this section, firstly, the moves are looked at in order and if it is determined that there is an error, the error is printed and the next move of the same player is made. This process continues until no error occurs. Also in this part is the ending mechanism of the game.

Programmer's Catalogue

1) Imports, Reading Files and Exceptions

```
import sys
```

The data entered with this code is taken from the interface.

```
try:
    try:
        player1Datas = open(sys.argv[1],"r")
    except IOError:
        print("IOError: cannot open " + sys.argv[1])
        outputFile.write("IOError: cannot open "+sys.argv[1])
        exit()

    try:
        player2Datas = open(sys.argv[2])
    except IOError:
        print("IOError: cannot open " + sys.argv[2])
        outputFile.write("IOError: cannot open " + sys.argv[2])
        exit()

    try:
        player2Moves = open(sys.argv[4])
    except IOError:
        print("IOError: cannot open " + sys.argv[4])
        outputFile.write("IOError: cannot open " + sys.argv[4])
        exit()

    try:
        player1Moves = open(sys.argv[3])
    except IOError:
        print("IOError: cannot open " + sys.argv[3])
        outputFile.write("IOError: cannot open " + sys.argv[3])
        exit()

except IndexError:
    print("IndexError: list index out of range")
    outputFile.write("IndexError: list index out of range")
    exit()
```

When the first try except block is entered less or more than the required index, it returns an error message and closes the program. The try except blocks inside check the correctness of the files in the entered index. If the files are wrong, it returns an error message and closes the program.

2) Creation of Empty Player Tables and Moves

```
player1Infos = open(sys.argv[1]).readlines()
player1List = []

player2Infos = open(sys.argv[2]).readlines()
player2List = []

for i in range(len(player1Infos)):
    takePlayer1Infos=player1Datas.readline().rstrip("\n").split(";")
    player1List.append(takePlayer1Infos)

for i in range(len(player1Infos)):
    takePlayer2Infos =player2Datas.readline().rstrip("\n").split(";")
    player2List.append(takePlayer2Infos)

player1MovesNumber = open(sys.argv[3]).readlines()
player2MovesNumber = open(sys.argv[4]).readlines()

takePlayer1Moves = player1Moves.readline().split(";")
takePlayer2Moves = player2Moves.readline().split(";")

round = 1

player1oyunHareket = []
player11 = []
for a in range(10):
    player11 = []
    for x in range(10):
        player11.append("-")
    player1oyunHareket.append(player11)

player2oyunHareket = []
player22 = []
for a in range(10):
    player22 = []
    for x in range(10):
        player22.append("-")
    player2oyunHareket.append(player22)
```

At first the .readlines command is used to read the size of the files. After looping as much as the contents of the file, each player's ships were located in order. Then the players move ";" parted with. Then, 10x10 empty lists of players were made to be shown in the table, and each coordinate was shown with "-".

3) Finding Ships

```
def playerShips(ship,shipInfo,dict):
    """The only purpose of this function is not to repeat the same things as
the function I used to find ships."""
    harfler = "ABCDEFGHIJ"
    ship = shipInfo.readlines()
    for i in range(len(ship)):
        isim = ship[i][0:2]
        satır = ship[i].split(":")[1].split(",")[0]
        sütun = ship[i].split(":")[1].split(",")[1][0]
        taraf = ship[i].split(";")[1]
        sütunHarfi = harfler.index(sütun)
        numberDict = {"A": 0, "B": 1, "C": 2, "D": 3, "E": 4, "F": 5, "G": 6,
"H": 7, "I": 8, "J": 9}

        if isim[0] == "B":
            if taraf == "right":
                dict[isim] = [harfler[sütunHarfi] + satır, harfler[sütunHarfi
+ 1] + satır,
                                      harfler[sütunHarfi + 2] + satır,
harfler[sütunHarfi + 3] + satır]

            elif taraf == "down":
                dict[isim] = [sütun + satır, sütun + str(int(satır) + 1),
sütun + str(int(satır) + 2),
                                      sütun + str(int(satır) + 3)]

        elif isim[0] == "P":
            if taraf == "right":
                dict[isim] = [harfler[sütunHarfi] + satır, harfler[sütunHarfi
+ 1] + satır]


            elif taraf == "down":
                dict[isim] = [sütun + satır, sütun + str(int(satır) + 1)]
```

In this function, ships named B and P were looked at. The contents of the supplied OptionalPlayer.txt files were read and the locations of ships B and P were found. According to the given file, the ships were determined to the right or down, and the positions of the ships were assigned to a dictionary.

```
def player1Ships():
    """this function is to find ships."""
    global ships1Dict,player1Ships,player1ShipsInfos
    for i in range(10):
        if "C" in player1List[i]:
            indexOfC = player1List[i].index("C")
            if player1List[i][indexOfC + 1] == "C":
                ships1Dict["C"] = [harfler[indexOfC] + str(i + 1),
harfler[indexOfC + 1] + str(i + 1),
                                   harfler[indexOfC + 2]
                                   + str(i + 1), harfler[indexOfC + 3] +
str(i + 1), harfler[indexOfC + 4] + str(i + 1)]
            else:
                ships1Dict["C"].append(harfler[indexOfC] + str(i + 1))

        if "S" in player1List[i]:
            indexOfS = player1List[i].index("S")
            if player1List[i + 1][indexOfS] == "S":
                ships1Dict["S"].append(harfler[indexOfS] + str(i + 1))

            else:
                ships1Dict["S"] = [harfler[indexOfS] + str(i + 1),
harfler[indexOfS + 1] + str(i + 1),
                                   harfler[indexOfS + 2] + str(i + 1)]

        if "D" in player1List[i]:
            indexOfD = player1List[i].index("D")
            try:
                if player1List[i + 1][indexOfD] == "D":
                    ships1Dict["D"] = [harfler[indexOfD] + str(i),
harfler[indexOfD] + str(i + 1),
                                       harfler[indexOfD] + str(i + 2)]
            except:
                pass
            try:
                if player1List[i][indexOfD + 1] == "D":
                    ships1Dict["D"] = [harfler[indexOfD] + str(i + 1),
harfler[indexOfD + 1] + str(i + 1),
                                       harfler[indexOfD + 2] + str(i +
1)]
            except:
```

In the 2nd part of finding ships, the player1 function has been created for the special cases of player1 ships. In this function, player 1's "C", "S" and "D" ships are tried to be found. Possible error situations are evaluated and the next move is checked. Finding ships briefly, first the first position of any ship is found, then the index on the right of that ship is checked, if there is the same ship there, these ships are next to each other and their ships are recorded. If there's no ship to your right, that means it's below. At that time, ships are selected to the lower side and their positions are recorded.

```
def player2Ships():
    """this function is to find ships."""
    global ships2Dict
    harfler = "ABCDEFGHIJ"
    player2Ships = player2ShipsInfos.readlines()
    for i in range(10):
        if "C" in player2List[i]:
            indexOfC = player2List[i].index("C")
            if player2List[i][indexOfC + 1] == "C":
                ships2Dict["C"] = [harfler[indexOfC] + str(i + 1),
harfler[indexOfC + 1] + str(i + 1),
                                   harfler[indexOfC + 2]
                                   + str(i + 1), harfler[indexOfC + 3] +
str(i + 1), harfler[indexOfC + 4] + str(i + 1)]
            else:
                ships2Dict["C"].append(harfler[indexOfC] + str(i + 1))

        if "S" in player2List[i]:
            indexOfS = player2List[i].index("S")
            if player2List[i][indexOfS] == "S":
                ships2Dict["S"].append(harfler[indexOfS] + str(i + 1))

            else:
                ships2Dict["S"] = [harfler[indexOfC] + str(i + 1),
harfler[indexOfC + 1] + str(i + 1),
                                   harfler[indexOfC + 2] + str(i + 1)]

        if "D" in player2List[i]:
            indexOfD = player2List[i].index("D")
            try:
                if player2List[i + 1][indexOfD] == "D":
                    ships2Dict["D"] = [harfler[indexOfD] + str(i),
harfler[indexOfD] + str(i + 1),
                                       harfler[indexOfD] + str(i + 2)]

            except:
                ships2Dict["D"].append(harfler[indexOfD] + str(i + 1))
```

In the 3rd part of finding ships, the player2 function has been created for the special cases of player2 ships. In this function, the "C", "S" and "D" ships of the 2nd player are tried to be found. Possible error situations are evaluated and the next move is checked. In order to find the ships briefly, first the first position of any ship is found, then the index to the right of that ship is checked, and if there is the same ship, these ships are next to each other and their ships are recorded. If there is no ship on your right, it means below. At that time, the ships are selected at the bottom and their positions are recorded.

4) Ship Appearances

```
def shipsApperance(dict,position,patrol):
    """this function checks sinking ships and checks sinking chart"""
    if dict["B1"][0] == "-" and dict["B1"][1] == "-" and dict["B1"][2]
== "-" and dict["B1"][
        3] == "-":
        position["B1"] = "X "

    if dict["B2"][0] == "-" and dict["B2"][1] == "-" and dict["B2"][2]
== "-" and dict["B2"][
        3] == "-":
        position["B2"] = "X "

    if dict["C"][0] == "-" and dict["C"][1] == "-" and dict["C"][2] ==
"-" and dict["C"][
        3] == "-" and ships2Dict["C"][4] == "-":
        position["C"] = "X "

    if dict["D"][0] == "-" and dict["D"][1] == "-" and dict["D"][2] ==
"-":
        position["D"] = "X"

    if dict["S"][0] == "-" and dict["S"][1] == "-" and dict["S"][2] ==
"-":
        position["S"] = "X"

    if dict["P1"][0] == "-" and dict["P1"][1] == "-":
        position["P1"] = "X"
        patrol["P1"] = "X"

    if dict["P2"][0] == "-" and dict["P2"][1] == "-":
        position["P2"] = "X"
        patrol["P2"] = "X"

    if dict["P3"][0] == "-" and dict["P3"][1] == "-":
        position["P3"] = "X"
        patrol["P3"] = "X"

    if dict["P4"][0] == "-" and dict["P4"][1] == "-":
        position["P4"] = "X"
        patrol["P4"] = "X"
```

In this function, it is checked whether the ships were hit or not. It is checked according to the ship names in the dictionary of ships. For example, the ship named "B1" has 4 indexes and all indexes are checked. If the 4 index is hit, that is, "X", then "B1" = "X", meaning it was hit, is given in the other variable where that ship is located. This process is common for both players in their ships.

5) Printing of Ships

```
def printShipsName():
    """this function is for printing the status of ships"""
    number1=0
    number2 = 0
    ships= ["P1","P2","P3","P4"]
    for i in range(len(player1PatrolBoat)):
        if player1PatrolBoat[ships[i]] == "X":
            number1+=1
    for i in range(number1):
        patrolBoat1[i]= "X"

    for i in range(len(player2PatrolBoat)):
        if player2PatrolBoat[ships[i]] == "X":
            number2+=1
    for i in range(number2):
        patrolBoat2[i]= "X"

    print("\nCarrier\t      " + "%s\t\t\tCarrier       %s" %
(player1ShipsPosition["C"], player2ShipsPosition["C"]))
    print("Battleship    " + "%s%s\t\t\t" % (player1ShipsPosition["B1"],
player1ShipsPosition["B2"]) + "Battleship    %s %s" %
(player2ShipsPosition["B1"], player2ShipsPosition["B2"]))
    print("Destroyer     " + "%s\t\t\t" % player1ShipsPosition["D"] + "
Destroyer     %s" % player2ShipsPosition["D"])
    print("Submarine     " + "%s\t\t\t" % player1ShipsPosition["S"] + "
Submarine     %s" % player2ShipsPosition["S"])
    print("Patrol Boat   " + "%s %s %s %s\t\t" %
(patrolBoat1[0],patrolBoat1[1],patrolBoat1[2],patrolBoat1[3]) + "
Patrol Boat   %s %s %s %s" %
          (patrolBoat2[0],patrolBoat2[1],patrolBoat2[2],patrolBoat2[3]))

    outputFile.write("\nCarrier\t\t" + "%s\t\t\t\tCarrier\t\t%s" %
(player1ShipsPosition["C"], player2ShipsPosition["C"]))
    outputFile.write("\nBattleship" + "\t%s%s\t\t\t\t" %
(player1ShipsPosition["B1"], player1ShipsPosition["B2"]) +
"Battleship\t%s %s" % (player2ShipsPosition["B1"],
player2ShipsPosition["B2"]))
    outputFile.write(("\nDestroyer\t" + "%s\t\t\t\t" %
player1ShipsPosition["D"] + "Destroyer\t%s" %
player2ShipsPosition["D"]))
    outputFile.write(("\nSubmarine\t" + "%s\t\t\t\t" %
player1ShipsPosition["S"] + "Submarine\t%s" %
player2ShipsPosition["S"]))
    outputFile.write("\nPatrol Boat\t" + "%s %s %s %s\t\t\t" %
(patrolBoat1[0],patrolBoat1[1],patrolBoat1[2],patrolBoat1[3]) + "Patrol
Boat\t%s %s %s %s" %
(patrolBoat2[0],patrolBoat2[1],patrolBoat2[2],patrolBoat2[3]))
```

In this function, which prints ships both to the file and to the interface, the status of the ships is reviewed first. Because the ships may not have been hit in order. The status of the players' ships is then printed below their chart.

6) Making Tables

```
def makeTable():
    """this function is the table where I show the ships hit"""
    for row in range(10):
        if row == 9:
            print(str(row + 1), end="")
            outputFile.write(str(row + 1))
        else:
            print(str(row + 1), end=" ")
            outputFile.write(str(row + 1)+" ")
        for col in range(10):
            print(player1oyunHareket[row][col], end=" ")
            outputFile.write(player1oyunHareket[row][col]+" ")
        print("\t\t\t", end="")
        outputFile.write("\t\t")

        if row == 9:
            print(str(row + 1), end="")
            outputFile.write(str(row + 1))
        else:
            print(str(row + 1), end=" ")
            outputFile.write(str(row + 1)+" ")

        for col in range(10):
            print(player2oyunHareket[row][col], end=" ")
            outputFile.write(player2oyunHareket[row][col]+" ")
        print()
        outputFile.write("\n")
```

This function contains the main tables of the players. The control of the whole game is on this table, but this table is not the table that is printed on the screen. The controls of the whole game are made in such a way that the moves are not visible on the screen in this table.

7) Player's Moves

```
def player1Move(index):
    """this function makes the move of player 1"""
    global round, sungedShip2Number, player2ShipsPosition,
player1ShipsPosition, shipNameList
    print("Player1's Move\n")
    print("Round : " + str(round) + "\t\t\t\tGrid Size: 10x10\n")
    print("Player1's Hidden Board\t\t\tPlayer2's Hidden Board")
    print("  A B C D E F G H I J\t\t\t  A B C D E F G H I J")

    outputFile.write("\nPlayer1's Move\n")
    outputFile.write("\nRound : " + str(round) + "\t\t\t\tGrid Size:
10x10\n")
    outputFile.write("\nPlayer1's Hidden Board\t\tPlayer2's Hidden
Board\n")
    outputFile.write("  A B C D E F G H I J\t\t  A B C D E F G H I J\n")

    sayı = takePlayer1Moves[index].split(",")[0]
    harf = takePlayer1Moves[index].split(",")[1]

    makeTable()
    shipsApperance(ships2Dict,player2ShipsPosition,player2PatrolBoat)
    printShipsName()

    print("\nEnter your move: " + str(sayı) + "," + harf)
    print()
    outputFile.write("\n\nEnter your move: " + str(sayı) + "," +
harf+"\n")

    if player2List[int(sayı) - 1][harfler.index(harf)] == "":
        player2oyunHareket[int(sayı) - 1][harfler.index(harf)] = "O"
        player2List[int(sayı) - 1][harfler.index(harf)] = "O"
    else:
        player2oyunHareket[int(sayı) - 1][harfler.index(harf)] = "X"
        player2List[int(sayı) - 1][harfler.index(harf)] = "X"

    konum = harf + sayı

    shipNameList = ["B1", "B2", "P1", "P2", "P3", "P4", "C", "S", "D"]
    for i in range(len(ships2Dict)):
        try:
            shipName = ships2Dict[shipNameList[i]].index(konum)
        except:
            pass

        if konum in ships2Dict[shipNameList[i]]:
            ships2Dict[shipNameList[i]][shipName] = "-"
```

This function makes Player 1's move. First of all, information such as the number of moves, the owner of the move is printed. Next, the makeTable(), shipsApperancee(), printShipsName() functions we created earlier are called with the parameters. Then, according to the move, the sign in the table with "-" is made "X" if there is a ship, "O" if there is no ship. Finally, the condition of the ships is checked.

```
def player2Move(index):
    """this function makes the move of player 2"""
    global round, sungedShip1Number, player1ShipsPosition,
player2ShipsPosition, shipNameList
    print("Player2's Move\n")
    print("Round : " + str(round) + "\t\t\t\tGrid Size: 10x10\n")
    print("Player1's Hidden Board\t\t\tPlayer2's Hidden Board")
    print("  A B C D E F G H I J\t\t\t  A B C D E F G H I J")
    outputFile.write("\nPlayer2's Move\n")
    outputFile.write("\nRound : " + str(round) + "\t\t\t\tGrid Size:
10x10\n")
    outputFile.write("\nPlayer1's Hidden Board\t\tPlayer2's Hidden
Board\n")
    outputFile.write("  A B C D E F G H I J\t\t  A B C D E F G H I J\n")

    sayı = int(takePlayer2Moves[index].split(",")[0])
    harf = takePlayer2Moves[index].split(",")[1]
    makeTable()
    shipsApperance(ships1Dict, player1ShipsPosition,player1PatrolBoat)
    printShipsName()
    print("\nEnter your move: " + str(sayı) + "," + harf)
    print()
    outputFile.write("\n\nEnter your move: " + str(sayı) + "," +
harf+"\n")

    if player1List[sayı - 1][harfler.index(harf)] == "":
        player1oyunHareket[sayı - 1][harfler.index(harf)] = "O"
        player1List[sayı-1][harfler.index(harf)] = "O"
    else:
        player1oyunHareket[sayı - 1][harfler.index(harf)] = "X"
        player1List[sayı - 1][harfler.index(harf)] = "X"

    round += 1
    konum = harf + str(sayı)
    shipNameList = ["B1", "B2", "P1", "P2", "P3", "P4", "C", "S", "D"]
    for i in range(len(ships1Dict)):
        try:
            shipName = ships1Dict[shipNameList[i]].index(konum)
        except:
            pass

        if konum in ships1Dict[shipNameList[i]]:
            ships1Dict[shipNameList[i]][shipName] = "-"
```

This function makes Player 2's move. First of all, information such as the number of moves, the owner of the move is printed. Next, the makeTable(), shipsApperancee(), printShipsName() functions we created earlier are called with the parameters. Then, according to the move, the sign in the table with "-" is made "X" if there is a ship, "O" if there is no ship. Finally, the condition of the ships is checked.

8) Final Situation

```
def finalSituation(playerList1, playerList2):
    """this function shows the latest status"""
    print("Final Information\n")
    print("Player1's Board\t\t\tPlayer2's Board")
    print("  A B C D E F G H I J\t\t\t  A B C D E F G H I J")
    outputFile.write("\nFinal Information\n\n")
    outputFile.write("Player1's Board\t\t\tPlayer2's Board\n")
    outputFile.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 i in range(10):
        for j in range(10):
            if playerList2[i][j] == "":
                playerList2[i][j] = "-"

    for i in range(10):
        for j in range(10):
            if playerList1[i][j] == "":
                playerList1[i][j] = "-"

    for row in range(10):
        if row == 9:
            print(str(row + 1), end="")
            outputFile.write(str(row + 1))
        else:
            print(str(row + 1), end=" ")
            outputFile.write(str(row + 1)+" ")
        for col in range(10):
            print(playerList1[row][col], end=" ")
            outputFile.write(playerList1[row][col]+" ")
        print("\t\t\t", end="")
        outputFile.write("\t\t")
        if row == 9:
            print(str(row + 1), end="")
            outputFile.write(str(row + 1))
        else:
            print(str(row + 1), end=" ")
            outputFile.write(str(row + 1)+" ")

        for col in range(10):
            print(playerList2[row][col], end=" ")
            outputFile.write(playerList2[row][col]+" ")
        print()
        outputFile.write("\n")

    shipsApperance(ships1Dict, player1ShipsPosition,player1PatrolBoat)
    shipsApperance(ships2Dict, player2ShipsPosition,player2PatrolBoat)
    printShipsName()
```

This function displays the table with all moves played and the game over. First, he enters the final information. It then prints the table, but unlike the tables where the moves of the players are played, it also shows the ships of the winning player that were not hit. Therefore, it must be done differently from other tables. And at the end, it shows the positions of the ships again.

9) Game's Move Commands and Exceptions

```
for i in range(len(takePlayer1Moves)):
    harf="ABCDEFGHIJ"
    sungedShip1Number = 0

    try:
        if len(takePlayer1Moves[i])<3:
            raise IndexError
        if (int(takePlayer1Moves[i].split(",")[0]) > 10) or
(takePlayer1Moves[i].split(",")[1] not in harf):
            raise AssertionError
        player1Move(i)

    except AssertionError:
        print("AssertionError: Invalid Operation\n")
        outputFile.write("\nAssertionError: Invalid Operation\n")
        player1Move(i+1)

    except IndexError:
        print("IndexError: "+ takePlayer1Moves[i]+" something missing to
move\n")
        outputFile.write("\nIndexError: "+ takePlayer1Moves[i]+"
something missing to move\n")
        player1Move(i + 1)

    except:
        print("ValueError " + takePlayer1Moves[i]+" is not a correct
move\n")
        outputFile.write("\nValueError " + takePlayer1Moves[i]+" is not
a correct move\n")
        player1Move(i + 1)

    try:
        if len(takePlayer2Moves[i])<3:
            raise IndexError

        if (int(takePlayer2Moves[i].split(",")[0]) > 10) or
(takePlayer2Moves[i].split(",")[1] not in harf):
            raise AssertionError
        player2Move(i)

    except AssertionError:
        print("AssertionError: Invalid Operation\n")
        outputFile.write("\nAssertionError: Invalid Operation\n")
        player2Move(i + 1)

    except IndexError:
        print("IndexError: "+ takePlayer2Moves[i]+" something missing to
move\n")
        outputFile.write("\nIndexError: "+ takePlayer2Moves[i]+"
something missing to move\n")
        player2Move(i + 1)

    except:
        print("ValueError " + takePlayer2Moves[i] + " is not a correct
move\n")
        outputFile.write("\nValueError " + takePlayer2Moves[i] + " is
not a correct move\n")
        player2Move(i + 1)
```

First of all, these last commands control the start and end of the game, that is, the whole mechanism. Each move is first checked for errors. First, IndexError is thrown if the move is less than 3 indexes. Then, the error of entering the wrong move is checked with AssertionError, such as the move choosing 15 rows as in B15. Apart from these, if there is an unnecessary error in the move (like "A,A"), ValueError is returned. These errors are checked separately within the player as well.

```
for i in range(len(ships2Dict)):
    sungedShip1Number += ships1Dict[shipNameList[i]].count("-")

sungedShip2Number = 0
for i in range(len(ships2Dict)):
    sungedShip2Number += ships2Dict[shipNameList[i]].count("-")

if sungedShip2Number == 27:
    player2Move(i)
    if sungedShip2Number == 27:
        print("DRAW\n")
        outputFile.write("\nDRAW\n")
        finalSituation(player1List,player2List)
        break
    print("Player1 Wins!\n")
    outputFile.write("\nPlayer1 Wins!\n")
    finalSituation(player1List, player2List)
    break

if sungedShip1Number == 27:
    print("Player2 Wins!\n")
    outputFile.write("\nPlayer2 Wins!\n")
    finalSituation(player1List, player2List)
    break
```

Finally, there is the ending mechanism of the game, which is the continuation of the above piece of code. Here, the completion status is checked by reading the information in the list indicating whether the ships have sunk before. Also, if the first player wins the game first, the second player is given an extra 1 right. Checked if the game ends in a draw if the player destroys player1's last ship with two moves.

User Catalogue

Sample Run

>python3 Assignment4.py “Player1.txt” “Player2.txt” “Player1.in” “Player2.in”

Some output examples:

```
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

Player2’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 - - - - O - - - - -
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: 1,J
```

{…}

{More rounds continues} {…}

```
Player1's Move

Round : 59              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 O O - O - O X - O O     1 - - - - - - - O - - -
2 - - O O - O X - O -     2 - - X X - X X O O X
3 - - - - - X O - X X -   3 - O O - O O O - - X
4 O - - O - - X O - O     4 X O X - O O O - - O
5 O O O - X O - - - O     5 - - - - O O X X - X
6 - X X X X - O O O O     6 O O O - - - O O - -
7 O O - O O - - X O O     7 O - - O - X - O O -
8 O - - O O O O - O X     8 - - - O O O O X O -
9 - - - - X - O O - X     9 - X O X X O - X O -
10O - - O O - O - O -      10O X O O O - O O O O

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

Enter your move: 4,I

Player2's Move

Round : 59              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 O O - O - O X - O O     1 - - - - - - - O - - -
2 - - O O - O X - O -     2 - - X X - X X O O X
3 - - - - - X O - X X -   3 - O O - O O O - - X
4 O - - O - - X O - O     4 X O X - O O O - O O
5 O O O - X O - - - O     5 - - - - O O X X - X
6 - X X X X - O O O O     6 O O O - - - O O - -
7 O O - O O - - X O O     7 O - - O - X - O O -
8 O - - O O O O - O X     8 - - - O O O O X O -
9 - - - - X - O O - X     9 - X O X X O - X O -
10O - - O O - O - O -      10O X O O O - O O O O

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

Enter your move: 9,F
```

```
Player2's Move

Round : 83              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 O O O O - O X - O O     1 - - O O O O O - O X
2 - O O O - O X O O O     2 - O X X X X X O O X
3 - X - O X O X X X O     3 - O O O O O O O O X
4 O X - O X O X O - O     4 X O X X O O O O O O
5 O O O O X O X O - O     5 - - O - O O X X - X
6 - X X X X - O O O O     6 O O O O - - O O O -
7 O O O O O X X X O O     7 O X O O - X O O O O
8 O O - O O O O - O X     8 O - - O O O O X O O
9 - - O - X X O O O X     9 - X O X X O O X O -
10O X X O O - O - O X     10O X O O O O O O O O

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

Enter your move: 2,E

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     1 - - O O O O O - O X
2 - O O O X O X O O O     2 D O X X X X X O O X
3 - X - O X O X X X O     3 D O O O O O O O O X
4 O X - O X O X O - O     4 X O X X O O O O O O
5 O O O O X O X O - O     5 - - O - O O X X B X
6 - X X X X - O O O O     6 O O O O - - O O O -
7 O O O O O X X X O O     7 O X O O P X O O O O
8 O O - O O O O - O X     8 O B - O O O O X O O
9 - - O - X X O O O X     9 - X O X X O O X O -
```

Grading Table

Evaluation	Points	Evaluate Yourself/ Guess Grading
Readable Codes and Meaningful Naming	5	4
Using Explanatory Comments	5	5
Efficiency(avoiding unnecessary actions)	5	4
Function Usage	15	15
Correctness, File I/O	30	30
Exceptions	20	20
Report	20	17
There are several negative evaluations	--	--
TOTAL = 95		