Hacettepe University

Department of Artificial Intelligence Engineering

BBM103 Assignment 4 Report

Bora Dere - 2220765021

01/01/2023

Analysis

Battleship is a well-known game in both board games and programming category. This game is played with 2 players and these players have their own ships in different amounts and types. Those types are generally Carrier, Battleship, Destroyer, Submarine and Patrol Boat. Different varieties of this game may contain different types and amounts of ships but base game mechanic is mostly the same. Both players have 2 boards: 1 is visible to both players and 1 is only visible to the owner. Players place their ship models to their hidden boards. Then 1st player makes their move and tries to shoot parts of the other player's ships. After that, 1st player marks their move to their visible board and the other player marks the same move to their hidden board. Then 2nd player makes their move and game keeps going. The end of the game also varies between different types of Battleship but most common one is announcing the player whom has all of their ships sunk as the loser side.

Design

Importing sys library and creating lists to use

First line imports the required library. *alphabet* list is to use while printing the boards. *used_letters* list is to use at checking operations later.

```
import sys
alphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
used_letters = ["C", "B", "P", "S", "D"]
```

fix_number function

This function takes 1 argument. It checks the length of the number given. It basically does an exception for number 10 while printing table, since its length is different than the others.

```
def fix_number(num):
    return str(num) if len(str(num)) == 2 else str(num) + " "
```

What it does:

1. Take an argument *num*.
2. Check its length.
3. If its length is 2 (*num* is 10) return *num* itself.
4. Else, return *num* + a whitespace.

txt_organizer function

This function takes 1 argument. It reformats the given the files which have .in extension to make their usage easier.

```
def txt_organizer(data):
    liste = []
    for j in data:
        liste.append([i if i else "-" for i in j.split(";")])
    return liste
```

printer_and_writer function

This function takes 2 arguments since it prints and writes two lists simultaneously. Its main job is to create required table format and make them aligned.

```
def printer_and_writer(list1, list2):
    print("   " + " ".join(alphabet) + "\t\t" + "   " + " ".join(alphabet))
    print("\n".join([fix_number(i + 1) + " ".join(list1[i]) + "\t\t" +
                     fix_number(i + 1) + " ".join(list2[i]) for i in range(len(list1))]))
    f.write("   " + " ".join(alphabet) + "\t\t" + "   " + " ".join(alphabet) + "\n")
    f.write("\n".join([fix_number(i + 1) + " ".join(list1[i]) + "\t\t" +
                       fix_number(i + 1) + " ".join(list2[i]) for i in range(len(list1))]) + "\n")
```

ship_recorder1 function

This function takes 4 arguments. Those are the list which will be checked, dictionary of C ship, dictionary of S ship and dictionary of D ship, respectively. This function checks the list point by point starting from top left corner.

```
def ship_recorder1(liste, dict_c, dict_s, dict_d):
    for i in range(len(liste)):
        for j in range(len(liste)):
            if liste[i][j] in used_letters:
                if liste[i][j] == "C":
                    temp = {alphabet[j] + str(i + 1): "C"}
                    dict_c.update(temp)
                if liste[i][j] == "S":
                    temp = {alphabet[j] + str(i + 1): "S"}
                    dict_s.update(temp)
                if liste[i][j] == "D":
                    temp = {alphabet[j] + str(i + 1): "D"}
                    dict_d.update(temp)
```

What it does:

1. Check every single point starting from top left corner, then moving rightwards. When end of the line is reached, jump to the next line.
2. If it is in *used_letters* list, add its location to its corresponding dictionary.

ship_recorder2 function

This function takes 3 arguments which are optional file, ship name and ship dictionary, respectively. It is implemented to use the optional files.

```
def ship_recorder2(optional, n, dicti):
    for i in optional:
        if i.startswith(n):
            y_i = i.replace(n + ":", "").split(",")[1][0]
            x_i = i.replace(n + ":", "").split(",")[0]
            index = alphabet.index(y_i)
            if "B" in n:
                if "right" in i:
                    for j in range(4):
                        temp = {alphabet[index + j] + x_i: n[0]}
                        dicti.update(temp)
                if "down" in i:
                    for j in range(4):
                        temp = {y_i + str(int(x_i) + j): n[0]}
                        dicti.update(temp)
            if "P" in n:
                if "right" in i:
                    for j in range(2):
                        temp = {alphabet[index + j] + x_i: n[0]}
                        dicti.update(temp)
                if "down" in i:
                    for j in range(2):
                        temp = {y_i + str(int(x_i) + j): n[0]}
                        dicti.update(temp)
```

What it does:

1. Obtain the y and x coordinate of the given starting point of the ship separately.
2. Convert the y coordinate to its corresponding number index, since it is a letter at first.
3. Filter the given ship name regarding its type and orientation.
4. Record the ship locations based on the results of the filtering process on 3rd step.

shooter function

This function takes 4 arguments. They are: moves list, actual list, hidden list and dictionary list since shooting process has an affect on those lists. It contains some exception handling in itself.

```
def shooter(moves, actual, hidden, dict_list):
    try:
        row = moves[i - 1].split(",")[0]
        column = moves[i - 1].split(",")[1]
        assert int(row) < 11
        assert column < "K"
        move = column + row
        if actual[int(row) - 1][alphabet.index(column)] == "-":
            actual[int(row) - 1][alphabet.index(column)] = "O"
            hidden[int(row) - 1][alphabet.index(column)] = "O"
        else:
            actual[int(row) - 1][alphabet.index(column)] = "X"
            hidden[int(row) - 1][alphabet.index(column)] = "X"
        for d in dict_list:
            if move in d:
                d.pop(move)
    except ValueError:
        f.write(f"ValueError: Your move {moves_1[i - 1]} does not fit the supported move pattern. "
                f"Supported last move is: 10,J\n\n")
    except AssertionError:
        f.write("AssertionError: Invalid Operation.\n\n")
    except IndexError:
        f.write(f"IndexError: Your move '{moves_1[i - 1]}' does not fit the supported move pattern. "
                f"Supported move pattern is: 'number,letter' while number is less than 11 and positive, "
                f"and letter comes before than 'K' in alphabet.\n\n")
```

What it does:

1. To make it fit to the dictionary format, obtain the row and column information separately.
2. Assert that row is less than 11 and column's index in alphabet is less than K's index.
3. Define a *move* variable.
4. Check whether given move hits a ship.
5. If it hits a ship, mark that place with X.
6. Else, mark it with O.
7. If given move is in one of the dictionaries which is in *dict_list*, pop it.

8. If code raises ValueError, state that given move does not fit the supported move pattern and print the last supported move. So, player will understand that and will not provide a move which is located at left or above of the last supported move.
9. If code raises AssertionError, state that it is an invalid operation.
10. If code raises IndexError, state that given move does not fit the supported move pattern and print the last supported move. Also explain the supported move pattern so player will not provide faulty moves such as “K,K”, “A,11”, “2,2” anymore.

ship_status_printer_and_writer function

This is by far the longest function and it takes no arguments. It is implemented so repeated printing and writing processes will not take space in code multiple times. Instead, this function will be called repeatedly.

```
def ship_status_printer_and_writer():
    global patrol1
    global patrol2
    print("Carrier\t\t-\t\t\t\tCarrier\t\t" if len(carrier1) != 0 else "Carrier\t\tX\t\t\t\tCarrier\t\t", end="")
    f.write("Carrier\t\t-\t\t\t\tCarrier\t\t" if len(carrier1) != 0 else "Carrier\t\tX\t\t\t\tCarrier\t\t")
    print("-" if len(carrier2) != 0 else "X")
    f.write("-" if len(carrier2) != 0 else "X")
    if len(battleship1_1) != 0 and len(battleship1_2) != 0:
        f.write("\nBattleship\t- -")
        print("Battleship\t- -", end="")
    if len(battleship1_1) == 0 and len(battleship1_2) != 0:
        f.write("\nBattleship\tX -")
        print("Battleship\tX -", end="")
    if len(battleship1_1) != 0 and len(battleship1_2) == 0:
        f.write("\nBattleship\tX -")
        print("Battleship\tX -", end="")
    if len(battleship1_1) == 0 and len(battleship1_2) == 0:
        f.write("\nBattleship\tX X")
        print("Battleship\tX X", end="")
    if len(battleship2_1) != 0 and len(battleship2_2) != 0:
        f.write("\t\t\t\tBattleship\t- -")
        print("\t\t\t\tBattleship\t- -")
    if len(battleship2_1) == 0 and len(battleship2_2) != 0:
        f.write("\t\t\t\tBattleship\tX -")
        print("\t\t\t\tBattleship\tX -")
    if len(battleship2_1) != 0 and len(battleship2_2) == 0:
        f.write("\t\t\t\tBattleship\tX -")
        print("\t\t\t\tBattleship\tX -")
    if len(battleship2_1) == 0 and len(battleship2_2) == 0:
        f.write("\t\t\t\tBattleship\tX X")
        print("\t\t\t\tBattleship\tX X")
```

Continued on the next page.

```
print("Destroyer\t-\t\t\t\tDestroyer\t" if len(destroyer1) != 0 else "Destroyer\tX\t\t\t\tDestroyer\t", end="")
f.write("\nDestroyer\t-\t\t\t\tDestroyer\t" if len(destroyer1) != 0 else "\nDestroyer\tX\t\t\t\tDestroyer\t")
print("-" if len(destroyer2) != 0 else "X")
f.write("-" if len(destroyer2) != 0 else "X")
print("Submarine\t-\t\t\t\tSubmarine\t" if len(submarine1) != 0 else "Submarine\tX\t\t\t\tSubmarine\t", end="")
f.write("\nSubmarine\t-\t\t\t\tSubmarine\t" if len(submarine1) != 0 else "\nSubmarine\tX\t\t\t\tSubmarine\t")
print("-" if len(submarine2) != 0 else "X")
f.write("-" if len(submarine2) != 0 else "X")
```

```
if patrol1 > 6:
    f.write("\nPatrol Boat\t- - - -")
    print("Patrol Boat\t- - - -", end="")
if 6 >= patrol1 > 4:
    f.write("\nPatrol Boat\tX - - -")
    print("Patrol Boat\tX - - -", end="")
if 4 >= patrol1 > 2:
    f.write("\nPatrol Boat\tX X - -")
    print("Patrol Boat\tX X - -", end="")
if 2 >= patrol1 > 0:
    f.write("\nPatrol Boat\tX X X -")
    print("Patrol Boat\tX X X -", end="")
if patrol1 == 0:
    f.write("\nPatrol Boat\tX X X X")
    print("Patrol Boat\tX X X X", end="")
if patrol2 > 6:
    f.write("\t\t\tPatrol Boat\t- - - -")
    print("\t\t\tPatrol Boat\t- - - -")
if 6 >= patrol2 > 4:
    f.write("\t\t\tPatrol Boat\tX - - -")
    print("\t\t\tPatrol Boat\tX - - -")
if 4 >= patrol2 > 2:
    f.write("\t\t\tPatrol Boat\tX X - -")
    print("\t\t\tPatrol Boat\tX X - -")
if 2 >= patrol2 > 0:
    f.write("\t\t\tPatrol Boat\tX X X -")
    print("\t\t\tPatrol Boat\tX X X -")
if patrol2 == 0:
    f.write("\t\t\tPatrol Boat\tX X X X")
    print("\t\t\tPatrol Boat\tX X X X")
```

What it does:

1. Print the sunk and floating ship information with respect to required format.

Main *try* block and codes below it

Since it is the longest portion of this assignment, it will be explained partially.

```
with open(sys.argv[1], "r") as f:
    data_1 = f.read().splitlines()
with open(sys.argv[2], "r") as f:
    data_2 = f.read().splitlines()
with open(sys.argv[3], "r") as f:
    moves_1 = f.read().split(";")
moves_1.remove("")
with open(sys.argv[4], "r") as f:
    moves_2 = f.read().split(";")
moves_2.remove("")
with open("OptionalPlayer1.txt", "r") as f:
    optional_data1 = f.read().splitlines()
with open("OptionalPlayer2.txt", "r") as f:
    optional_data2 = f.read().splitlines()
f = open("Battleship.out", "w")
```

What it does:

1. Open the given files to reading and doing required split processes to make them easy to use.
2. Open (and create if not existing) a output file with the requested name and writing type.

```
hidden_1 = [["-" for j in range(10)] for i in range(10)]
hidden_2 = [["-" for j in range(10)] for i in range(10)]
```

What it does:

1. Create the hidden (which does not show the ship information) lists in 2D array format.

```
#Ships for the first player
carrier1 = {}
destroyer1 = {}
submarine1 = {}
battleship1_1 = {}
battleship1_2 = {}
patrol1_1 = {}
patrol1_2 = {}
patrol1_3 = {}
patrol1_4 = {}

#Ships for the second player
carrier2 = {}
destroyer2 = {}
submarine2 = {}
battleship2_1 = {}
battleship2_2 = {}
patrol2_1 = {}
patrol2_2 = {}
patrol2_3 = {}
patrol2_4 = {}
```

What it does:

1. Create a dictionary for each ship of each player, so length checking based on the ship type will be easier.

```
dict_list1 = [carrier1, destroyer1, submarine1, battleship1_1, battleship1_2, patrol1_1, patrol1_2, patrol1_3, patrol1_4]
dict_list2 = [carrier2, destroyer2, submarine2, battleship2_1, battleship2_2, patrol2_1, patrol2_2, patrol2_3, patrol2_4]
```

What it does:

1. Gathering all ships of both players in separate dictionary lists to use later iterating on.

```
actual_1 = txt_organizer(data_1)
actual_2 = txt_organizer(data_2)
ship_recorder1(actual_1, carrier1, submarine1, destroyer1)
ship_recorder1(actual_2, carrier2, submarine2, destroyer2)
ship_recorder2(optional_data1, "B1", battleship1_1)
ship_recorder2(optional_data1, "B2", battleship1_2)
ship_recorder2(optional_data1, "P1", patrol1_1)
ship_recorder2(optional_data1, "P2", patrol1_2)
ship_recorder2(optional_data1, "P3", patrol1_3)
ship_recorder2(optional_data1, "P4", patrol1_4)
ship_recorder2(optional_data2, "B1", battleship2_1)
ship_recorder2(optional_data2, "B2", battleship2_2)
ship_recorder2(optional_data2, "P1", patrol2_1)
ship_recorder2(optional_data2, "P2", patrol2_2)
ship_recorder2(optional_data2, "P3", patrol2_3)
ship_recorder2(optional_data2, "P4", patrol2_4)
```

What it does:

1. This block does the list and dictionary creating processes. It calls necessary functions multiple times to do the job.

Driver Code

This block of code does the main printing and writing processes and calls the functions when they are required to keep the game running.

```
if len(moves_1) == len(moves_2):
    print("Battle of Ships Game\n")
    f.write("Battle of Ships Game\n\n")
    for i in range(1, len(moves_1) + 1):
        patrol1 = len(patrol1_2) + len(patrol1_2) + len(patrol1_3) + len(patrol1_4)
        patrol2 = len(patrol2_2) + len(patrol2_2) + len(patrol2_3) + len(patrol2_4)
        print("Player1's Move\n")
        f.write("Player1's Move\n\n")
        print("Round :", str(i) + "\t\t\t\t\tGrid Size: 10x10\n")
        f.write("Round :" + " " + str(i) + "\t\t\t\t\tGrid Size: 10x10\n\n")
```

Continued on the next page.

```
    print("Player1's Hidden Board\t\tPlayer2's Hidden Board")
    f.write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n")
    printer_and_writer(hidden_1, hidden_2)
    print("")  #newline
    f.write("\n")
    ship_status_printer_and_writer()
    print("")
    f.write("\n\n")
    print("Enter your move:", moves_1[i - 1] + "\n")
    f.write("Enter your move:" + " " + moves_1[i - 1] + "\n\n")
    shooter(moves_1, actual_2, hidden_2, dict_list2)
    print("Player2's Move\n")
    f.write("Player2's Move\n\n")
    print("Round :", str(i) + "\t\t\t\t\tGrid Size: 10x10\n")
    f.write("Round :" + " " + str(i) + "\t\t\t\t\tGrid Size: 10x10\n\n")
    print("Player1's Hidden Board\t\tPlayer2's Hidden Board")
    f.write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n")
    printer_and_writer(hidden_1, hidden_2)
    print("")  #newline
    f.write("\n")
    ship_status_printer_and_writer()
    print("")
    f.write("\n\n")
    print("Enter your move:", moves_2[i - 1] + "\n\n")
    f.write("Enter your move:" + " " + moves_2[i - 1] + "\n\n")
    shooter(moves_2, actual_1, hidden_1, dict_list1)
```

What this portion does:

1. Check whether the length of both of the move files are equal.
2. State that game begins.
3. Calculate the total length of each players' patrol boats.
4. Do the printing and writing processes and call the required functions whenever they are needed to achieve the requested output format.

```
length1 = len(carrier1) + len(battleship1_1) + len(battleship1_2) + len(destroyer1) + len(submarine1) +\
          len(patrol1_1) + len(patrol1_2) + len(patrol1_3) + len(patrol1_4)
length2 = len(carrier2) + len(battleship2_1) + len(battleship2_2) + len(destroyer2) + len(submarine2) +\
          len(patrol2_1) + len(patrol2_2) + len(patrol2_3) + len(patrol2_4)
```

What this portion does:

1. Calculate the total length of the ship dictionaries for each player. This will be useful at the decision-making stage.

```
    if length1 > length2:
        print("Player1 Wins!\n")
        f.write("Player1 Wins!\n\n")
    if length2 > length1:
        print("Player2 Wins!\n")
        f.write("Player2 Wins!\n\n")
    if length2 == length1:
        print("It is a Draw!\n")
        f.write("It is a Draw!\n\n")
    print("Final Information\n")
    f.write("Final Information\n\n")
    print("Player1's Board\t\t\t\tPlayer2's Board")
    f.write("Player1's Board\t\t\t\tPlayer2's Board\n")
    printer_and_writer(actual_1, actual_2)
    print("")
    f.write("\n")
    ship_status_printer_and_writer()
else:
    f.write("kaBOOM: run for your life!")
```

What this portion does:

1. Compare the lengths which were calculated right before.
2. Decide which player has won based on the results of the comparison.
3. State that game has ended and print the final information of the tables and ship information.
4. If different amounts of moves are provided, go kaBOOM since it is an error.

Exception case

```
except FileNotFoundError:
    f = open("Battleship.out", "w")
    checklist = []
    if sys.argv[1] != "Player1.txt":
        checklist.append(sys.argv[1])
    if sys.argv[2] != "Player2.txt":
        checklist.append(sys.argv[2])
    if sys.argv[3] != "Player1.in":
        checklist.append(sys.argv[3])
    if sys.argv[4] != "Player2.in":
        checklist.append(sys.argv[4])
    f.write("IOError: input file(s) " + ", ".join(i for i in checklist) + " is/are not reachable.")
except IndexError:
    f = open("Battleship.out", "w")
    f.write("kaBOOM: run for your life!")
```

What it does:

1. If main scope gives FileNotFoundError, check the command provided to terminal.
2. Gather the faulty arguments to a list.
3. State that the code gives an error, IOError. And print the faulty arguments, so user can realize which ones are faulty and can change them.
4. If main scope gives IndexError, state that an error has occurred. Since this exception is not covered in PDF, it gives kaBOOM error.

Programmer's Catalogue

Understanding the assignment was difficult for me because at first, I thought we would write a game which takes real-time user input. After couple of reading, I realized it was not the case. Still, I had some difficulties and my beginning to code was delayed because I wanted to wait for you to share the text files. But you did not share them at the time that I hoped you would, so I started really slow to code.

Output format checking was also a different problem. You will see that on Python and Developer Server, some of the outputs look unaligned. But it is completely fine on text files, even though it uses same number of tabs. Since you said we can ignore them on Python or server if text files are correct, (which should be the case because it is the same output, there is actually no reason for them to be unaligned) I ignored them.

Since writing mode "write" is used, output will be cleared after every single run. So, no extra cleaning is needed. Whole code will be shared part by part, since it is too long to fit in a single page.

```
import sys
#importing the required module
alphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
#letter list to use on tables
used_letters = ["C", "B", "P", "S", "D"]
#used letter list to use while checking


#basically making an exception for number 10 in list
def fix_number(num):
    return str(num) if len(str(num)) == 2 else str(num) + " "


#function to organize the .in files
def txt_organizer(data):
    liste = []
    for j in data:
        liste.append([i if i else "-" for i in j.split(";")])
    return liste


#function to print and write the lists
def printer_and_writer(list1, list2):
    print("   " + " ".join(alphabet) + "\t\t" + "   " + " ".join(alphabet))
    print("\n".join([fix_number(i + 1) + " ".join(list1[i]) + "\t\t" +
fix_number(i + 1) + " ".join(list2[i]) for i in range(len(list1))]))
    f.write("   " + " ".join(alphabet) + "\t\t" + "   " + " ".join(alphabet)
+ "\n")
```

```
    f.write("\n".join([fix_number(i + 1) + " ".join(list1[i]) + "\t\t" +
fix_number(i + 1) + " ".join(list2[i]) for i in range(len(list1))]) +
"\n")


#function to record the ship data to actual lists and dictionaries
def ship_recorder1(liste, dict_c, dict_s, dict_d):
    for i in range(len(liste)):
        for j in range(len(liste)):
            if liste[i][j] in used_letters:
                if liste[i][j] == "C":
                    temp = {alphabet[j] + str(i + 1): "C"}
                    dict_c.update(temp)
                if liste[i][j] == "S":
                    temp = {alphabet[j] + str(i + 1): "S"}
                    dict_s.update(temp)
                if liste[i][j] == "D":
                    temp = {alphabet[j] + str(i + 1): "D"}
                    dict_d.update(temp)


#function to record the ship data with optional files
def ship_recorder2(optional, n, dicti):
    for i in optional:
        if i.startswith(n):
            y_i = i.replace(n + ":", "").split(",")[1][0]
            x_i = i.replace(n + ":", "").split(",")[0]
            index = alphabet.index(y_i)
            if "B" in n:
                if "right" in i:
                    for j in range(4):
                        temp = {alphabet[index + j] + x_i: n[0]}
                        dicti.update(temp)
                if "down" in i:
                    for j in range(4):
                        temp = {y_i + str(int(x_i) + j): n[0]}
                        dicti.update(temp)
            if "P" in n:
                if "right" in i:
                    for j in range(2):
                        temp = {alphabet[index + j] + x_i: n[0]}
                        dicti.update(temp)
                if "down" in i:
                    for j in range(2):
                        temp = {y_i + str(int(x_i) + j): n[0]}
                        dicti.update(temp)


#function to execute the specified shooting processes
def shooter(moves, actual, hidden, dict_list):
    try:
        row = moves[i - 1].split(",")[0]
        column = moves[i - 1].split(",")[1]
        assert int(row) < 11
```

```
        assert column < "K"
        move = column + row
        if actual[int(row) - 1][alphabet.index(column)] == "-":
            actual[int(row) - 1][alphabet.index(column)] = "O"
            hidden[int(row) - 1][alphabet.index(column)] = "O"
        else:
            actual[int(row) - 1][alphabet.index(column)] = "X"
            hidden[int(row) - 1][alphabet.index(column)] = "X"
        for d in dict_list:
            if move in d:
                d.pop(move)
    except ValueError:
        f.write(f"ValueError: Your move {moves_1[i - 1]} does not fit the
supported move pattern. "
                f"Supported last move is: 10,J\n\n")
    except AssertionError:
        f.write("AssertionError: Invalid Operation.\n\n")
    except IndexError:
        f.write(f"IndexError: Your move '{moves_1[i - 1]}' does not fit
the supported move pattern. Supported move pattern is: 'number,letter'
while number is less than 11 and positive, and letter comes before than
'K' in alphabet.\n\n")


#function to print and write the current ship status
def ship_status_printer_and_writer():
    global patrol1
    global patrol2
    print("Carrier\t\t-\t\t\t\tCarrier\t\t" if len(carrier1) != 0 else
"Carrier\t\tX\t\t\t\tCarrier\t\t", end="")
    f.write("Carrier\t\t-\t\t\t\tCarrier\t\t" if len(carrier1) != 0 else
"Carrier\t\tX\t\t\t\tCarrier\t\t")
    print("-" if len(carrier2) != 0 else "X")
    f.write("-" if len(carrier2) != 0 else "X")
    if len(battleship1_1) != 0 and len(battleship1_2) != 0:
        f.write("\nBattleship\t- -")
        print("Battleship\t- -", end="")
    if len(battleship1_1) == 0 and len(battleship1_2) != 0:
        f.write("\nBattleship\tX -")
        print("Battleship\tX -", end="")
    if len(battleship1_1) != 0 and len(battleship1_2) == 0:
        f.write("\nBattleship\tX -")
        print("Battleship\tX -", end="")
    if len(battleship1_1) == 0 and len(battleship1_2) == 0:
        f.write("\nBattleship\tX X")
        print("Battleship\tX X", end="")
    if len(battleship2_1) != 0 and len(battleship2_2) != 0:
        f.write("\t\t\t\tBattleship\t- -")
        print("\t\t\t\tBattleship\t- -")
    if len(battleship2_1) == 0 and len(battleship2_2) != 0:
        f.write("\t\t\t\tBattleship\tX -")
        print("\t\t\t\tBattleship\tX -")
    if len(battleship2_1) != 0 and len(battleship2_2) == 0:
        f.write("\t\t\t\tBattleship\tX -")
```

```
        print("\t\t\t\tBattleship\tX -")
    if len(battleship2_1) == 0 and len(battleship2_2) == 0:
        f.write("\t\t\t\tBattleship\tX X")
        print("\t\t\t\tBattleship\tX X")
    print("Destroyer\t-\t\t\t\tDestroyer\t" if len(destroyer1) != 0 else
"Destroyer\tX\t\t\t\tDestroyer\t", end="")
    f.write("\nDestroyer\t-\t\t\t\tDestroyer\t" if len(destroyer1) != 0
else "\nDestroyer\tX\t\t\t\tDestroyer\t")
    print("-" if len(destroyer2) != 0 else "X")
    f.write("-" if len(destroyer2) != 0 else "X")
    print("Submarine\t-\t\t\t\tSubmarine\t" if len(submarine1) != 0 else
"Submarine\tX\t\t\t\tSubmarine\t", end="")
    f.write("\nSubmarine\t-\t\t\t\tSubmarine\t" if len(submarine1) != 0
else "\nSubmarine\tX\t\t\t\tSubmarine\t")
    print("-" if len(submarine2) != 0 else "X")
    f.write("-" if len(submarine2) != 0 else "X")
    if patrol1 > 6:
        f.write("\nPatrol Boat\t- - - -")
        print("Patrol Boat\t- - - -", end="")
    if 6 >= patrol1 > 4:
        f.write("\nPatrol Boat\tX - - -")
        print("Patrol Boat\tX - - -", end="")
    if 4 >= patrol1 > 2:
        f.write("\nPatrol Boat\tX X - -")
        print("Patrol Boat\tX X - -", end="")
    if 2 >= patrol1 > 0:
        f.write("\nPatrol Boat\tX X X -")
        print("Patrol Boat\tX X X -", end="")
    if patrol1 == 0:
        f.write("\nPatrol Boat\tX X X X")
        print("Patrol Boat\tX X X X", end="")
    if patrol2 > 6:
        f.write("\t\t\tPatrol Boat\t- - - -")
        print("\t\t\tPatrol Boat\t- - - -")
    if 6 >= patrol2 > 4:
        f.write("\t\t\tPatrol Boat\tX - - -")
        print("\t\t\tPatrol Boat\tX - - -")
    if 4 >= patrol2 > 2:
        f.write("\t\t\tPatrol Boat\tX X - -")
        print("\t\t\tPatrol Boat\tX X - -")
    if 2 >= patrol2 > 0:
        f.write("\t\t\tPatrol Boat\tX X X -")
        print("\t\t\tPatrol Boat\tX X X -")
    if patrol2 == 0:
        f.write("\t\t\tPatrol Boat\tX X X X")
        print("\t\t\tPatrol Boat\tX X X X")


try:
    with open(sys.argv[1], "r") as f:
        data_1 = f.read().splitlines()
    with open(sys.argv[2], "r") as f:
        data_2 = f.read().splitlines()
    with open(sys.argv[3], "r") as f:
```

```
        moves_1 = f.read().split(";")
    moves_1.remove("")
    with open(sys.argv[4], "r") as f:
        moves_2 = f.read().split(";")
    moves_2.remove("")
    with open("OptionalPlayer1.txt", "r") as f:
        optional_data1 = f.read().splitlines()
    with open("OptionalPlayer2.txt", "r") as f:
        optional_data2 = f.read().splitlines()
    f = open("Battleship.out", "w")

    hidden_1 = [["-" for j in range(10)] for i in range(10)]
#the list which does not
    hidden_2 = [["-" for j in range(10)] for i in range(10)]
# show the ship information

    #Ships for the first player
    carrier1 = {}
    destroyer1 = {}
    submarine1 = {}
    battleship1_1 = {}
    battleship1_2 = {}
    patrol1_1 = {}
    patrol1_2 = {}
    patrol1_3 = {}
    patrol1_4 = {}

    #Ships for the second player
    carrier2 = {}
    destroyer2 = {}
    submarine2 = {}
    battleship2_1 = {}
    battleship2_2 = {}
    patrol2_1 = {}
    patrol2_2 = {}
    patrol2_3 = {}
    patrol2_4 = {}

    #Dictionary lists for iterating
    dict_list1 = [carrier1, destroyer1, submarine1, battleship1_1,
battleship1_2, patrol1_1, patrol1_2, patrol1_3, patrol1_4]
    dict_list2 = [carrier2, destroyer2, submarine2, battleship2_1,
battleship2_2, patrol2_1, patrol2_2, patrol2_3, patrol2_4]

    actual_1 = txt_organizer(data_1)
#creating the actual lists
    actual_2 = txt_organizer(data_2)
# which contain ship data
    ship_recorder1(actual_1, carrier1, submarine1, destroyer1)
    ship_recorder1(actual_2, carrier2, submarine2, destroyer2)
    ship_recorder2(optional_data1, "B1", battleship1_1)
    ship_recorder2(optional_data1, "B2", battleship1_2)
    ship_recorder2(optional_data1, "P1", patrol1_1)
    ship_recorder2(optional_data1, "P2", patrol1_2)
```

```
    ship_recorder2(optional_data1, "P3", patrol1_3)
    ship_recorder2(optional_data1, "P4", patrol1_4)
    ship_recorder2(optional_data2, "B1", battleship2_1)
    ship_recorder2(optional_data2, "B2", battleship2_2)
    ship_recorder2(optional_data2, "P1", patrol2_1)
    ship_recorder2(optional_data2, "P2", patrol2_2)
    ship_recorder2(optional_data2, "P3", patrol2_3)
    ship_recorder2(optional_data2, "P4", patrol2_4)

    #Driver Code which does the main printing and writing operations
    if len(moves_1) == len(moves_2):
        print("Battle of Ships Game\n")
        f.write("Battle of Ships Game\n\n")
        for i in range(1, len(moves_1) + 1):
            patrol1 = len(patrol1_2) + len(patrol1_2) + len(patrol1_3) +
len(patrol1_4)
            patrol2 = len(patrol2_2) + len(patrol2_2) + len(patrol2_3) +
len(patrol2_4)
            print("Player1's Move\n")
            f.write("Player1's Move\n\n")
            print("Round :", str(i) + "\t\t\t\t\tGrid Size: 10x10\n")
            f.write("Round :" + " " + str(i) + "\t\t\t\t\tGrid Size:
10x10\n\n")
            print("Player1's Hidden Board\t\tPlayer2's Hidden Board")
            f.write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n")
            printer_and_writer(hidden_1, hidden_2)
            print("")  #newline
            f.write("\n")
            ship_status_printer_and_writer()
            print("")
            f.write("\n\n")
            print("Enter your move:", moves_1[i - 1] + "\n")
            f.write("Enter your move:" + " " + moves_1[i - 1] + "\n\n")
            shooter(moves_1, actual_2, hidden_2, dict_list2)
            print("Player2's Move\n")
            f.write("Player2's Move\n\n")
            print("Round :", str(i) + "\t\t\t\t\tGrid Size: 10x10\n")
            f.write("Round :" + " " + str(i) + "\t\t\t\t\tGrid Size:
10x10\n\n")
            print("Player1's Hidden Board\t\tPlayer2's Hidden Board")
            f.write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n")
            printer_and_writer(hidden_1, hidden_2)
            print("")  #newline
            f.write("\n")
            ship_status_printer_and_writer()
            print("")
            f.write("\n\n")
            print("Enter your move:", moves_2[i - 1] + "\n\n")
            f.write("Enter your move:" + " " + moves_2[i - 1] + "\n\n")
            shooter(moves_2, actual_1, hidden_1, dict_list1)

        length1 = len(carrier1) + len(battleship1_1) + len(battleship1_2)
+ len(destroyer1) + len(submarine1) + len(patrol1_1) + len(patrol1_2) +
len(patrol1_3) + len(patrol1_4)
```

```
        length2 = len(carrier2) + len(battleship2_1) + len(battleship2_2)
+ len(destroyer2) + len(submarine2) + len(patrol2_1) + len(patrol2_2) +
len(patrol2_3) + len(patrol2_4)

        if length1 > length2:
            print("Player1 Wins!\n")
            f.write("Player1 Wins!\n\n")
        if length2 > length1:
            print("Player2 Wins!\n")
            f.write("Player2 Wins!\n\n")
        if length2 == length1:
            print("It is a Draw!\n")
            f.write("It is a Draw!\n\n")
        print("Final Information\n")
        f.write("Final Information\n\n")
        print("Player1's Board\t\t\t\tPlayer2's Board")
        f.write("Player1's Board\t\t\t\tPlayer2's Board\n")
        printer_and_writer(actual_1, actual_2)
        print("")
        f.write("\n")
        ship_status_printer_and_writer()
    else:
        f.write("kaBOOM: run for your life!")

except FileNotFoundError:
    f = open("Battleship.out", "w")
    checklist = []
    if sys.argv[1] != "Player1.txt":
        checklist.append(sys.argv[1])
    if sys.argv[2] != "Player2.txt":
        checklist.append(sys.argv[2])
    if sys.argv[3] != "Player1.in":
        checklist.append(sys.argv[3])
    if sys.argv[4] != "Player2.in":
        checklist.append(sys.argv[4])
    f.write("IOError: input file(s) " + ", ".join(i for i in checklist) +
" is/are not reachable.")
except IndexError:
    f = open("Battleship.out", "w")
    f.write("kaBOOM: run for your life!")
#Bora Dere, 2220765021
```

User Catalogue

Usage of *fix_number* function

It takes one integer argument and checks its length. If it is one character long, returns it with a whitespace at the end of it. If it is two characters long, returns the number itself.

Example: fix_number(5)

Usage of *txt_organizer* function

It takes one argument and organizes given file with .in extension.

Example: txt_organizer(data_1)

Usage of *printer_and_writer* function

It takes two list arguments since it prints two lists simultaneously.

Example: printer_and_writer(hidden_1, hidden_2)

Usage of *ship_recorder1* function

It takes four arguments. First one is the list which will be checked. Other 3 are dictionaries separated by their ship types. It checks all points in the given list and records the specified ships.

Example: ship_recorder1(actual_1, carrier1, submarine1, destroyer1)

Usage of *ship_recorder2* function

It takes three arguments. First one is the optional ship list, second one is the ship type and name and the last one is the corresponding ship dictionary. It uses OptionalPlayer text files to record the ship locations.

Example: ship_recorder2(optional_data2, "P1", patrol2_1)

Usage of *shooter* function

It takes four arguments. First one is the used move list. Second, third and the fourth ones are the actual and hidden list and the dictionary list which get affected by the shooting process, respectively.

Example: shooter(moves_1, actual_2, hidden_2, dict_list2)

Usage of *ship_status_printer_and_writer* function

It takes no arguments, so it is by far the easiest function to use. It does bulk printing and writing processes and prevents repeating those big lines of code multiple times.

Example: ship_status_printer_and_writer()

Restrictions

When a player tries to make a faulty move, game states that that move is not supported. But instead of waiting for that player to provide a new move, it passes the turn to the other player. It is not what you said in PDF but what you said does not suit the nature of this game. Let's assume that both players have 15 moves played and we will process them. Also assume 1st player provided 10 faulty moves and those are the first 10 moves. What you wanted is to wait for the 11th move of the first player but it ruins the logic of this game. It needs to be played one by one, each player following the other. In your way 2nd player plays 11 moves consecutively and at the end of the game, plays more moves than the first player, which are just not good for the sake of the game. So, my game will state that 1st player provided a faulty move and pass the turn to the other player. This way, the game will be played turn by turn and the order will not be ruined.

Assume each player provided 50 moves but all of the 1st player's ships sink after 30 rounds played. The game still processes those extra 20 moves and decides the final result after that. Because in this assignment, we process a game which actually was concluded even before its beginning. The game is not played simultaneously by two players and it is not a real-time game. In normal conditions, game would end and players could not make another move after it's ended. But this assignment just prints the output of a game which already has been played and concluded. So, providing an input file as described in the assumption is just wrong and it is against this assignment's nature. So, this situation is not considered and frankly, should not be expected to be considered as explained above.

Requested output format can be obtained in text files but it becomes unaligned (for some reason) on PyCharm terminal and Developer Server. Here are some examples of the output file (it has 2 more lines than the output with no errors because an error occurs midway of the given input):

```
Player1's Move

Round : 19 ---------- 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 - - X - O O ----------> 1 - - - - - - - - - -
2 - - - - - - - X - - ----------> 2 - - - - - - X - O - -
3 - - - - - - - - - - ----------> 3 - - - - - - - O - - -
4 O - - - - - - - - - ----------> 4 X - X - O O - - - - -
5 O - - - - - O - - - O ----------> 5 - - - - - - - X - - -
6 - - - - - X - - - O - ----------> 6 - - - - - - - O - - -
7 O - - - - - - - - - O ----------> 7 O - - - - - - - - - -
8 O - - - - - O - - O - ----------> 8 - - - - - - O - - O -
9 - - - - - - - - - - ----------> 9 - - - O - - - - - - -
10- - - O O - - - - - ----------> 10- - - - - O - O O O -

Carrier ----------> - ----------> Carrier ----------> -
Battleship ----------> - - ----------> Battleship ----------> - -
Destroyer ----------> - ----------> Destroyer ----------> -
Submarine ----------> - ----------> Submarine ----------> -
Patrol Boat ----------> - - - - ----------> Patrol Boat ----------> - - - - -

Enter your move: 8,G

Player2's Move

Round : 19 ---------- 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 - - X - O O ----------> 1 - - - - - - - - - -
2 - - - - - - - X - - ----------> 2 - - - - - - X - O - -
3 - - - - - - - - - - ----------> 3 - - - - - - - O - - -
4 O - - - - - - - - - ----------> 4 X - X - O O - - - - -
5 O - - - - - O - - - O ----------> 5 - - - - - - - X - - -
6 - - - - - X - - - O - ----------> 6 - - - - - - - O - - -
7 O - - - - - - - - - O ----------> 7 O - - - - - - - - - -
8 O - - - - - O - - O - ----------> 8 - - - - - - O O - O -
9 - - - - - - - - - - ----------> 9 - - - O - - - - - - -
10- - - O O - - - - - ----------> 10- - - - - O - O O O -

Carrier ----------> - ----------> Carrier ----------> -
Battleship ----------> - - ----------> Battleship ----------> - -
Destroyer ----------> - ----------> Destroyer ----------> -
Submarine ----------> - ----------> Submarine ----------> -
Patrol Boat ----------> - - - - ----------> Patrol Boat ----------> - - - - -

Enter your move: 6,B
```

```
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 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 -
10 O X X O O - O - O X          10 O X O O O O O O O O

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

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**