Hacettepe University

Department Of Artificial Intelligence

BBM 103 Assignment 4 Report

Yusuf Emir Cömert – 2220765023

05.01.2023

Contents

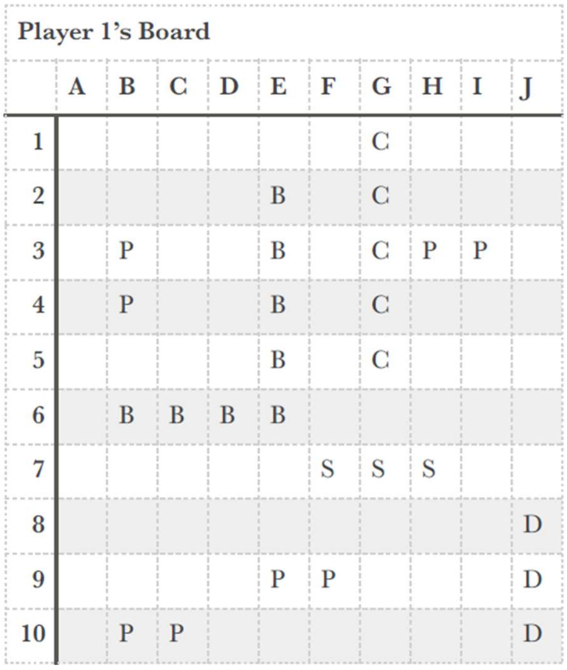
Battle of Ships

1 – Analysis:

In this assignment, it is requested us to code a python program of a Battle Of Ships Game which we have 2 inputs per players and there has to 2 players named Player 1 and Player 2.

Each player should place their ships somewhere at a 10x10 map and attack some squares randomly. When player hits another player's any ships, there will be put "x" to the board; if that shoot is a miss, there will be put an "O (Big Oh letter)" to the board. Each player has their hidden board, and they will be empty at the start of the game. In every move board would updating that's why output is 4175 lines. If there is a wrong move at the input file, the round would continue with the other player's move (which teacher told us not to do but I couldn't fix how to do it).

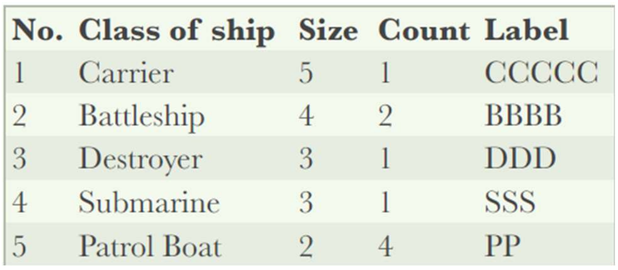
For my example input files, ships are in these positions:



Player 1's Board

	A	B	C	D	E	F	G	H	I	J
1							C			
2					B		C			
3		P			B		C	P	P	
4		P			B		C			
5					B		C			
6		B	B	B	B					
7						S	S	S		
8										D
9					P	P				D
10		P	P							D

And this is the meanings of the letters:



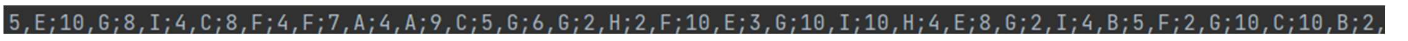
No.	Class of ship	Size	Count	Label
1	Carrier	5	1	CCCCC
2	Battleship	4	2	BBBB
3	Destroyer	3	1	DDD
4	Submarine	3	1	SSS
5	Patrol Boat	2	4	PP

Inputs should've given like "Player1.txt", "Player2.txt", "Player1.in" , "Player2.in". And the output file is like "Battleship.out" file. You don't have to put names exactly because I opened file with writing mode which means if you do not have this output file, program will create a new one for you.

".in" or "out" files are a generic file used by some spyware programs to track activity on a computer. This means that we can say that it is different type of txt files basically.

I have optional txt files which makes my job easier with detain groups of ships easier.

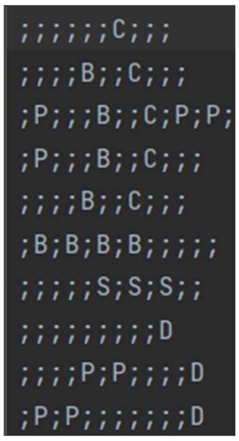
For example, inputs for Player1.in is the input file of the Player 1's attacking squares and so on.



```
5,E;10,G;8,I;4,C;8,F;4,F;7,A;4,A;9,C;5,G;6,G;2,H;2,F;10,E;3,G;10,I;10,H;4,E;8,G;2,I;4,B;5,F;2,G;10,C;10,B;2,
```

You can see that this is a part of my ".in" file. This means in the first round, player will attack 5-A square, and it goes like this. Commas separates the rows and columns, semicolons split the rounds.

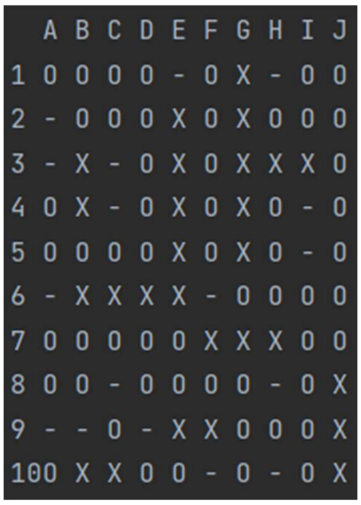
You can also see that what .txt files (Where the ships are located) looks like in the below.



```
;;;;;;C;;;
;;;;B;;C;;;
;P;;;B;;C;P;P;
;P;;;B;;C;;;
;;;;B;;C;;;
;B;B;B;B;;;;;
;;;;;S;S;S;;
;;;;;;;;;D
;;;;P;P;;;;D
;P;P;;;;;;;D
```

This shows that in the 1-G square is the part of a Carrier ship and so on. All inputs would be placed exactly like the board at the above.

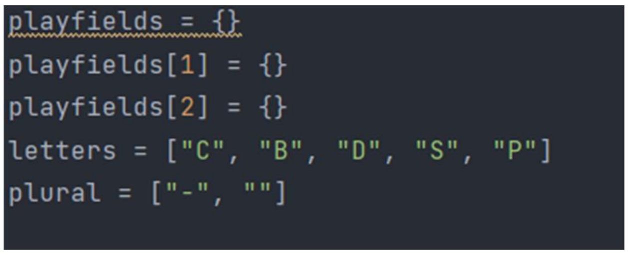
At the end of the game player 1's hidden board would look like at the bottom.



```
  A B C D E F G H I J
1 0 0 0 0 - 0 X - 0 0
2 - 0 0 0 X 0 X 0 0 0
3 - X - 0 X 0 X X X 0
4 0 X - 0 X 0 X 0 - 0
5 0 0 0 0 X 0 X 0 - 0
6 - X X X X - 0 0 0 0
7 0 0 0 0 0 X X X 0 0
8 0 0 - 0 0 0 0 - 0 X
9 - - 0 - X X 0 0 0 X
100 X X 0 0 - 0 - 0 X
```

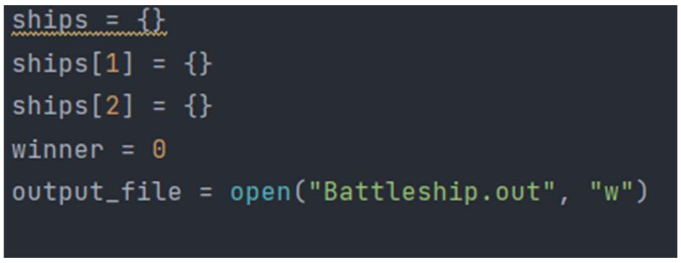
(Page Break)

2 - Design:



```
playfields = {}
playfields[1] = {}
playfields[2] = {}
letters = ["C", "B", "D", "S", "P"]
plural = ["-", ""]
```

Firstly I defined play fields and the letters (First letters of every ships). Playfield 1 and 2 means Player1’s Playfield and Player2’s Playfield.

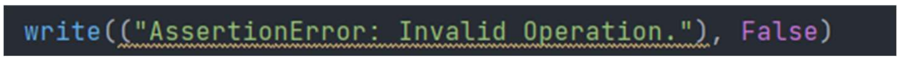


```
ships = {}
ships[1] = {}
ships[2] = {}
winner = 0
output_file = open("Battleship.out", "w")
```

Then I defined ships in a dictionary and there are ships 1 and ships 2 means Player1’ships and Player2’s ships.

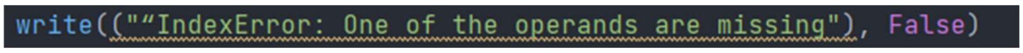
2.1 - Try-Excepts:

There are several try excepts for mistake moves and examples are this:



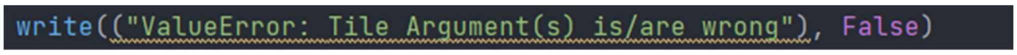
```
write(("AssertionError: Invalid Operation."), False)
```

If player gives input as 5:A ; Player has to separate columns and rows by column(,) and between two moves Player has to put semicolons(;). This causes error.



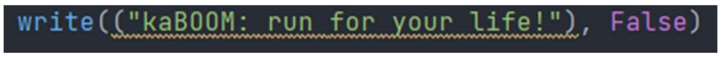
```
write((""“IndexError: One of the operands are missing"), False)
```

For example, if player inputs empty round the output will be like the picture shown at top.



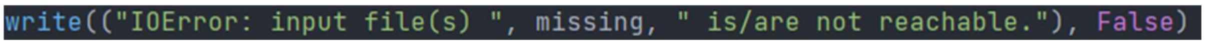
```
write(("ValueError: Tile Argument(s) is/are wrong"), False)
```

For example, when player inputs 5,5 tile instead of 5-A; There has to be an error because there is not a tile named 5,5.



```
write(("kaBOOM: run for your life!"), False)
```

This is basically the other exceptions that I might forgot.



```
write(("IOError: input file(s) ", missing, " is/are not reachable."), False)
```

Raises when input file(s) is/are missing. I will explain it at IO () function.

3-Programmer’s Catalogue:

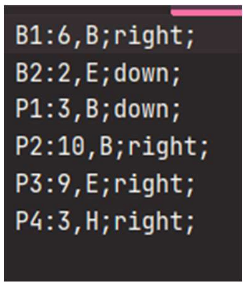
3.1 – Functions:

I used 10 functions this time and their names are “write, read, optional, readline, listship, orderlist, show, gameover, sink, io”.

As you can understand write function is for writing to terminal and writing for the .out file. In this assignment, it has been a bit complicated because end=”” made problem with writing.

Read function is the reading input files separately. For example, I have line1 which is the readlines of the first player, line2 is the readlines of the second player.

Optional is the longest function I had, and I used it for reading optional texts and making groups. Inside of the file is like this:



```
B1:6,B;right;
B2:2,E;down;
P1:3,B;down;
P2:10,B;right;
P3:9,E;right;
P4:3,H;right;
```

For example, this shows us that BattleShip1 ship’s position is starts from B-6 and goes right, BattleShip2 ship’s position is starts from E-2 and goes down.

Readline function is basically reading line by line, defining rows and columns etc.

ListShip function is not about table, but it is about keys of the tiles(squares). It is for making a tile “X” or “- “.

OrderList function is 2-line function. That is because solving a problem about printing the pattern which is at the bottom of the Board.

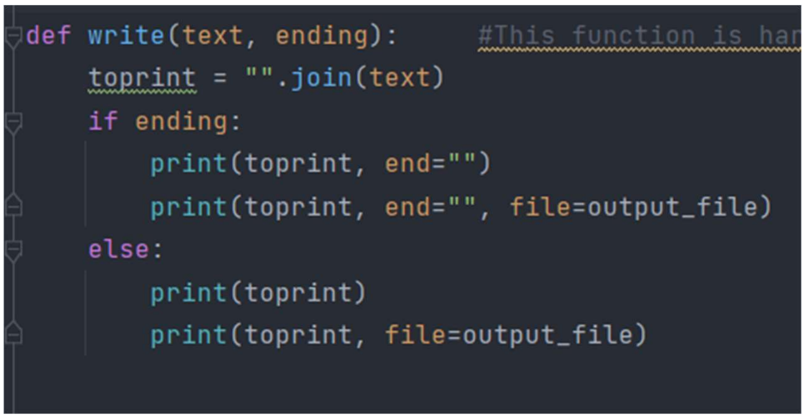
Show function is hard to explain but it is basically for printing the boards. Prints title of the boards, defines the ships and shows who is the winner (Player1, Player2 Wins or Draw situation).

I made Game over function because when I call this function somewhere, there is a parameter named breakgame, it is defaultly False. When I wanted to end the game, I make it True and game ends.

Sink function is for turn ship situations when ships sink. Splits the inputs and makes moves in this function. Some try excepts are in this function too.

IO function is checking any IO error like when one input is missing or something like that.

3.1.1 – Write () Function:

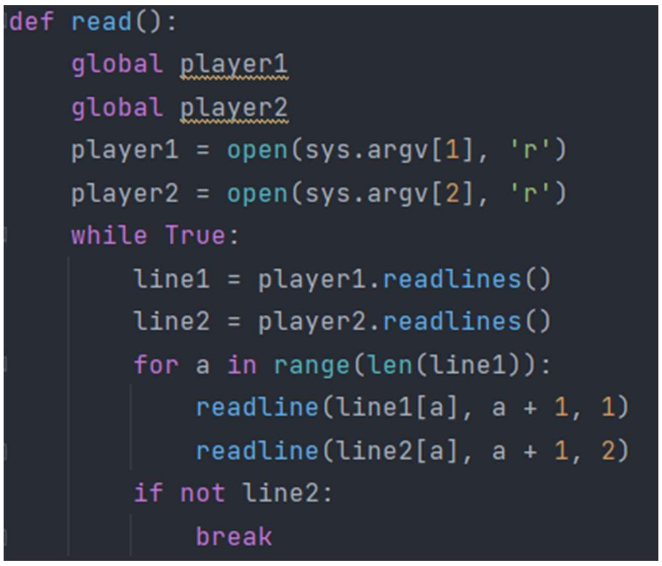


```python
def write(text, ending):      #This function is har
    toprint = "".join(text)
    if ending:
        print(toprint, end="")
        print(toprint, end="", file=output_file)
    else:
        print(toprint)
        print(toprint, file=output_file)
```

“’This function was easy to me but then it got an error. Error’s reason is that end = “” function is not working in this function. I must print output to both console and .out file so I had to make things change.’’’

As you can see write function works with two parameters. Firstly, we write a text and put comma, then write True or False. As I said end function was not working and causing error. So, if the parameter gives True, it puts end = “” to the end of the text. Else, just don’t do anything and print it to the output file.

3.1.2 – Read () Function:

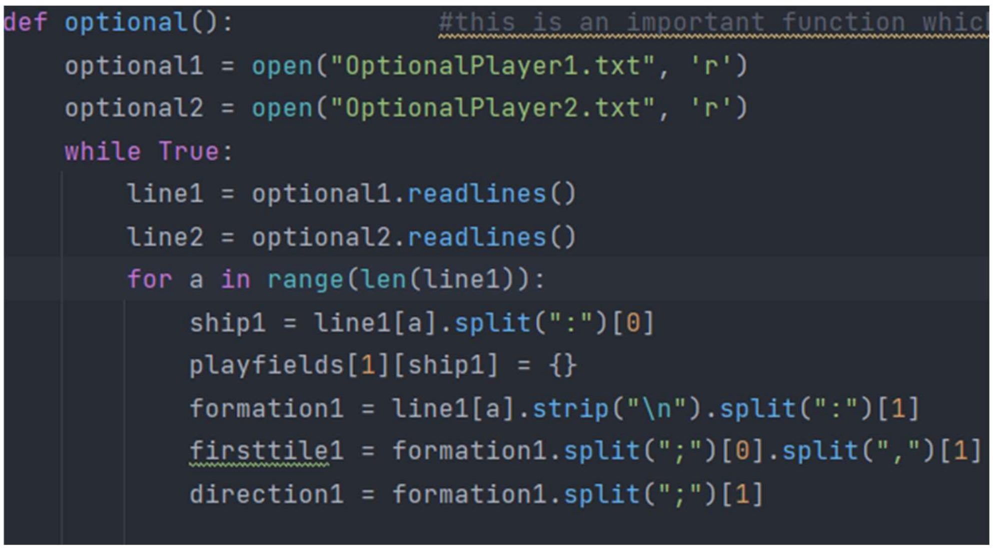


```python
def read():
    global player1
    global player2
    player1 = open(sys.argv[1], 'r')
    player2 = open(sys.argv[2], 'r')
    while True:
        line1 = player1.readlines()
        line2 = player2.readlines()
        for a in range(len(line1)):
            readline(line1[a], a + 1, 1)
            readline(line2[a], a + 1, 2)
        if not line2:
            break
```

This function takes sys.argv[] as inputs and reads line1(First Player's lines) and line2(Second Player's lines).There is a for loop and inside of it there are 2 readline command. It is because in this for loop a change and reads the lines line by line.

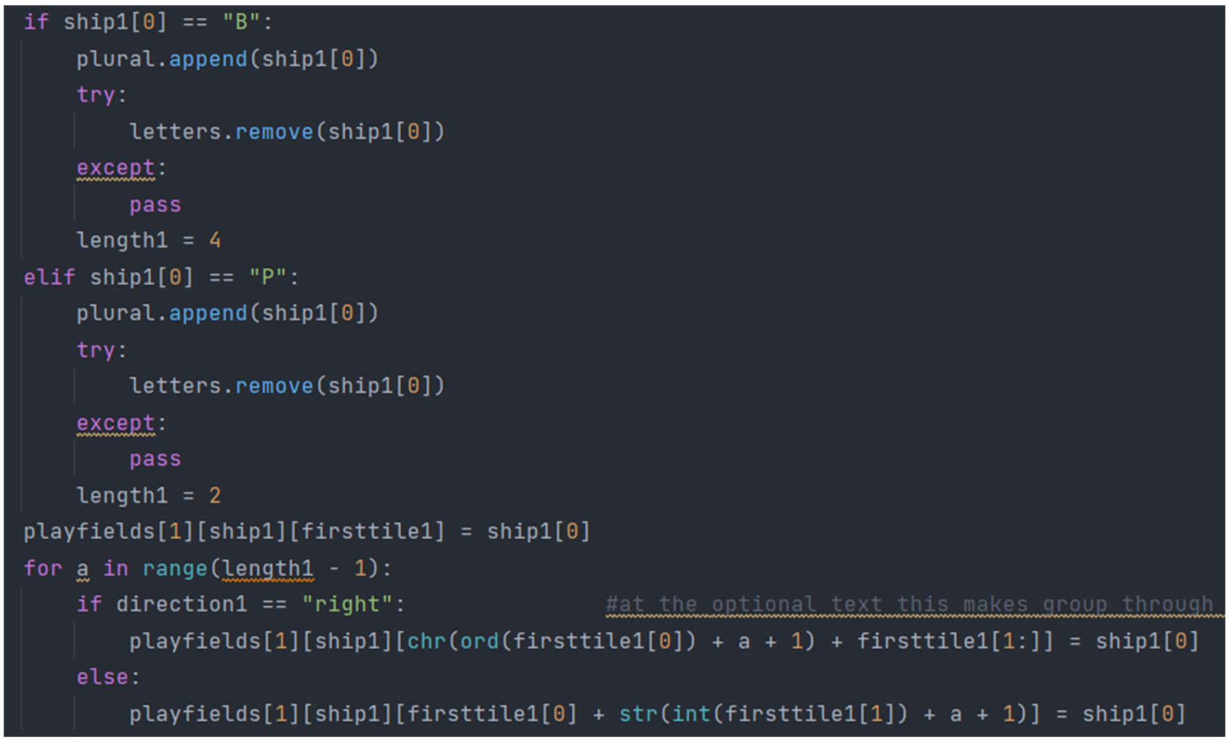
3.1.3 – Optional () function:

I mentioned about this function a bit and inside the optional text files, there are some informations about where the plural ships are.



```
def optional():             #this is an important function whic
    optional1 = open("OptionalPlayer1.txt", 'r')
    optional2 = open("OptionalPlayer2.txt", 'r')
    while True:
        line1 = optional1.readlines()
        line2 = optional2.readlines()
        for a in range(len(line1)):
            ship1 = line1[a].split(":")[0]
            playfields[1][ship1] = {}
            formation1 = line1[a].strip("\n").split(":")[1]
            firsttile1 = formation1.split(";")[0].split(",")[1]
            direction1 = formation1.split(";")[1]
```

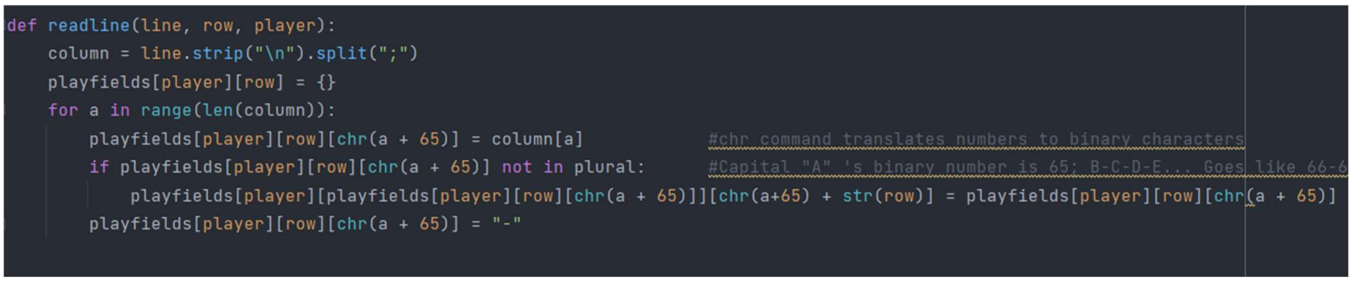
Firstly, I opened these files with reading mode, then defined the First player's lines and second's. You can see that I split and tried to get the values from text. Then I got the directions like right, left, down etc.



```
if ship1[0] == "B":
    plural.append(ship1[0])
    try:
        letters.remove(ship1[0])
    except:
        pass
    length1 = 4
elif ship1[0] == "P":
    plural.append(ship1[0])
    try:
        letters.remove(ship1[0])
    except:
        pass
    length1 = 2
playfields[1][ship1][firsttile1] = ship1[0]
for a in range(length1 - 1):
    if direction1 == "right":                    #at the optional text this makes group through
        playfields[1][ship1][chr(ord(firsttile1[0]) + a + 1) + firsttile1[1:]] = ship1[0]
    else:
        playfields[1][ship1][firsttile1[0] + str(int(firsttile1[1]) + a + 1)] = ship1[0]
```

Then I looked for the keys like Battleship's key is "B". Added them to a list. After that I did same thing to Patrol Boat. I defined their length as you can see. If direction is right, it will make the boats group and if one part of the boat sinks, boat would not have sink. All parts of the boats has to sink if player wants to make sink that ships. Then I did the same stuffs for the player 2. Codes are literally the same but their numbers. For example I put playfields [2] instead of 1.

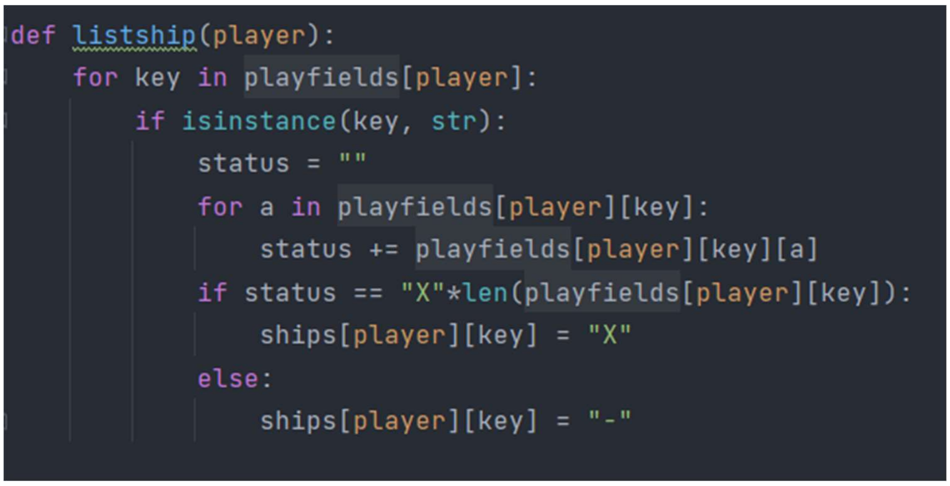
3.1.4 – Readline () Function:



```
def readline(line, row, player):
    column = line.strip("\n").split(";")
    playfields[player][row] = {}
    for a in range(len(column)):
        playfields[player][row][chr(a + 65)] = column[a]          #chr command translates numbers to binary characters
        if playfields[player][row][chr(a + 65)] not in plural:    #Capital "A" 's binary number is 65; B-C-D-E... Goes like 66-6
            playfields[player][playfields[player][row][chr(a + 65)]][chr(a+65) + str(row)] = playfields[player][row][chr(a + 65)]
        playfields[player][row][chr(a + 65)] = "-"
```

First, I defined the columns. Plural is a list which contains "- "and "" (empty). Chr function is a translator translates binary number to string. For example, capital A's binary number is 65, a is going to start being 0 and goes to length of the column. Then chr (a + 65) is going to be 65,66,67,68,69…. Their strings are A, B, C, D, E, F, G….

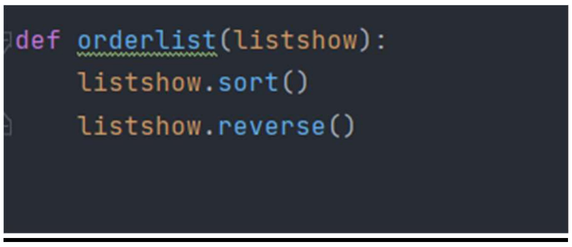
3.1.5 – ListShip () Function:



```
def listship(player):
    for key in playfields[player]:
        if isinstance(key, str):
            status = ""
            for a in playfields[player][key]:
                status += playfields[player][key][a]
            if status == "X"*len(playfields[player][key]):
                ships[player][key] = "X"
            else:
                ships[player][key] = "-"
```

I defined empty string named status. Then I made the keys “X” or “- “.

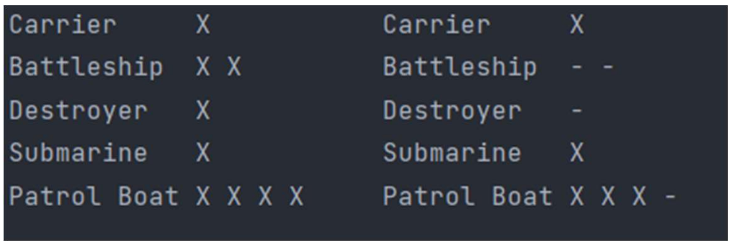
3.1.6 – OrderList() Function:



```
def orderlist(listshow):
    listshow.sort()
    listshow.reverse()
```

This was the easiest function and shortest. The logic of this function is correcting a mistake in the output. The mistake was this:

Under the boards, there are table shows which ships are sunk and which is not.



```
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 -
```

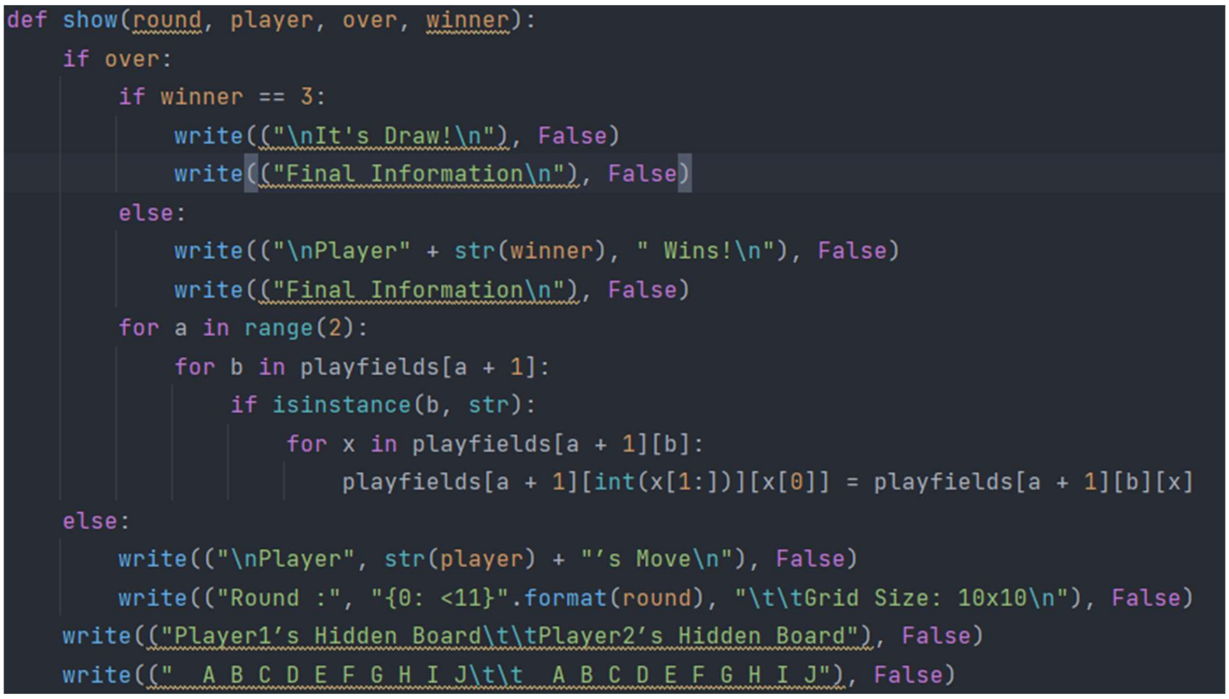
In this table when third patrol boat sinks, my program wrote “- - X – “

But I have to put “X“ to the beginning. I realized when I sorted this list where x and – are held.

Then I realized that "– "comes before "X" when we sort them. So, I reversed it, and the output is correct like "X - - - ".

3.1.7- Show () Function:

This is one of the longest function of mine. It is basically for boards.

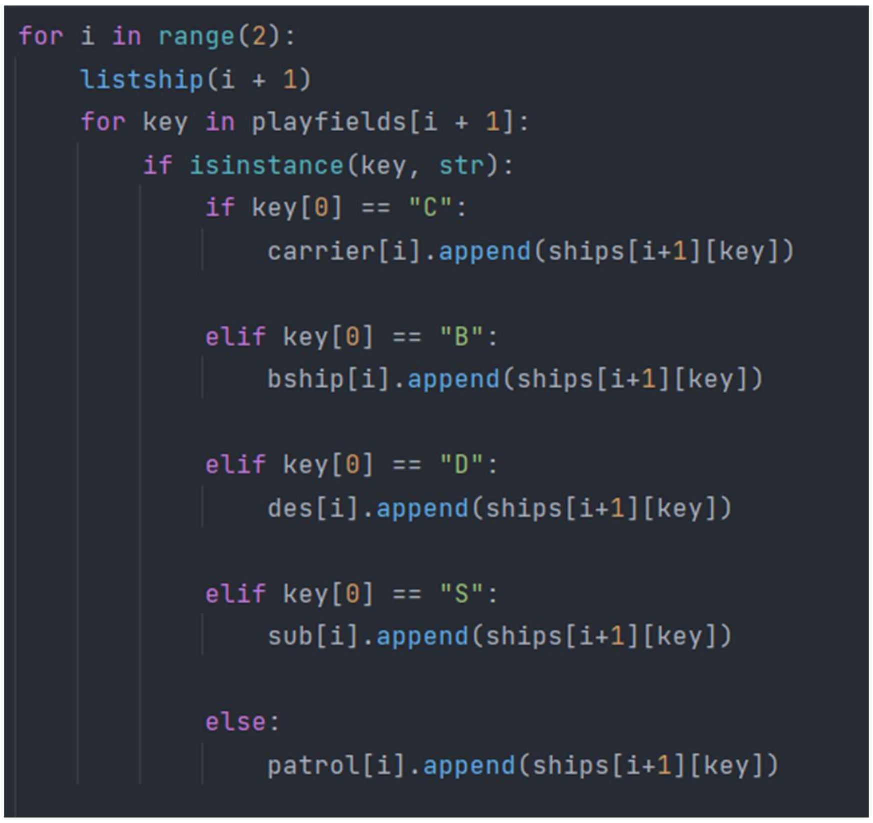


```python
def show(round, player, over, winner):
    if over:
        if winner == 3:
            write(("\nIt's Draw!\n"), False)
            write(("Final Information\n"), False)
        else:
            write(("\nPlayer" + str(winner), " Wins!\n"), False)
            write(("Final Information\n"), False)
        for a in range(2):
            for b in playfields[a + 1]:
                if isinstance(b, str):
                    for x in playfields[a + 1][b]:
                        playfields[a + 1][int(x[1:])][x[0]] = playfields[a + 1][b][x]
    else:
        write(("\nPlayer", str(player) + "'s Move\n"), False)
        write(("Round :", "{0: <11}".format(round), "\t\tGrid Size: 10x10\n"), False)
    write(("Player1's Hidden Board\t\tPlayer2's Hidden Board"), False)
    write(("  A B C D E F G H I J\t\t  A B C D E F G H I J"), False)
```

In every round program will check if the game is over or not. If it is over, writes who is the winner. If it is Draw, it prints draw.

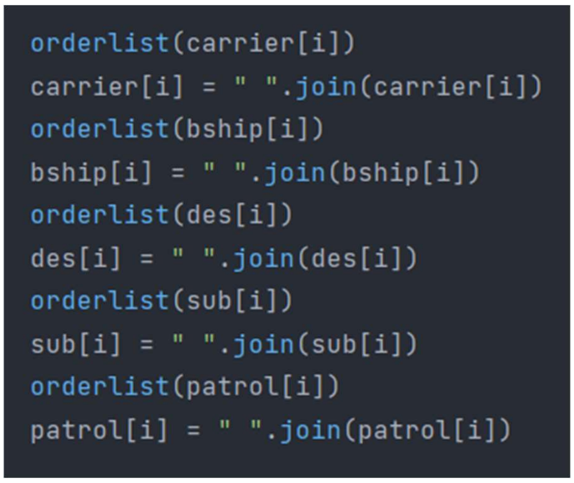
You can see that in the else section, it writes which player's move is this. At one line under it, there is a round information which gives us the number of the round.

There are both player's hidden boards and their row of the letters.



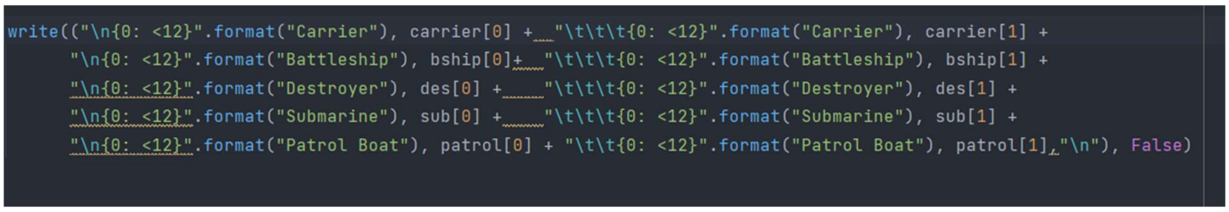
```python
for i in range(2):
    listship(i + 1)
    for key in playfields[i + 1]:
        if isinstance(key, str):
            if key[0] == "C":
                carrier[i].append(ships[i+1][key])

            elif key[0] == "B":
                bship[i].append(ships[i+1][key])

            elif key[0] == "D":
                des[i].append(ships[i+1][key])

            elif key[0] == "S":
                sub[i].append(ships[i+1][key])

            else:
                patrol[i].append(ships[i+1][key])
```

I basically just made when keys are B, C, D, S, P; I appended to their dictionary.



```python
orderlist(carrier[i])
carrier[i] = " ".join(carrier[i])
orderlist(bship[i])
bship[i] = " ".join(bship[i])
orderlist(des[i])
des[i] = " ".join(des[i])
orderlist(sub[i])
sub[i] = " ".join(sub[i])
orderlist(patrol[i])
patrol[i] = " ".join(patrol[i])
```

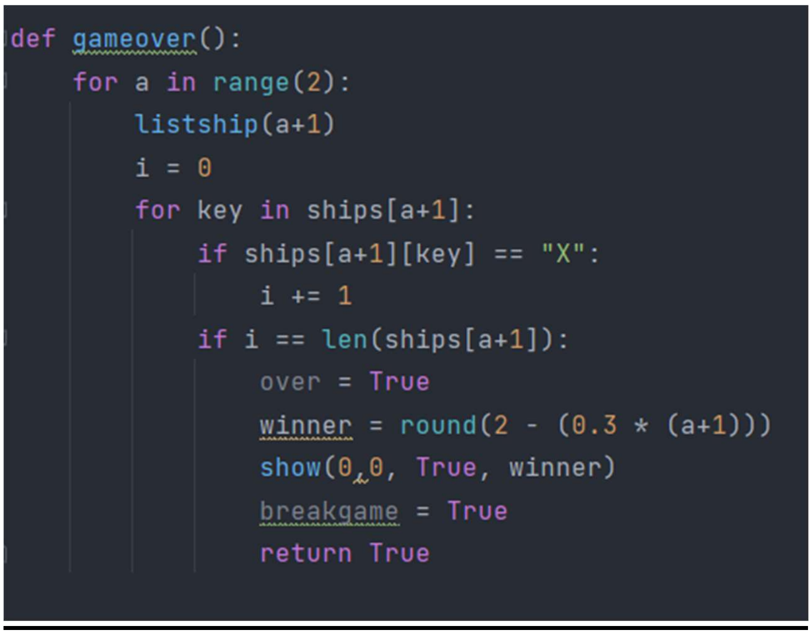
This is the reason that I made orderlist function. Every ship joins their empty strings.



```
write(("\n{0: <12}".format("Carrier"), carrier[0] + "\t\t\t{0: <12}".format("Carrier"), carrier[1] +
      "\n{0: <12}".format("Battleship"), bship[0]+ "\t\t\t{0: <12}".format("Battleship"), bship[1] +
      "\n{0: <12}".format("Destroyer"), des[0] + "\t\t\t{0: <12}".format("Destroyer"), des[1] +
      "\n{0: <12}".format("Submarine"), sub[0] + "\t\t\t{0: <12}".format("Submarine"), sub[1] +
      "\n{0: <12}".format("Patrol Boat"), patrol[0] + "\t\t{0: <12}".format("Patrol Boat"), patrol[1],"\n"), False)
```

This is how I wrote the table under the board (you can also see at eleventh page). This method called alignment, and this makes them look nice and not slipped. \n ‘s is as you can understand, to get to the next line.

3.1.8 – GameOver () Function :

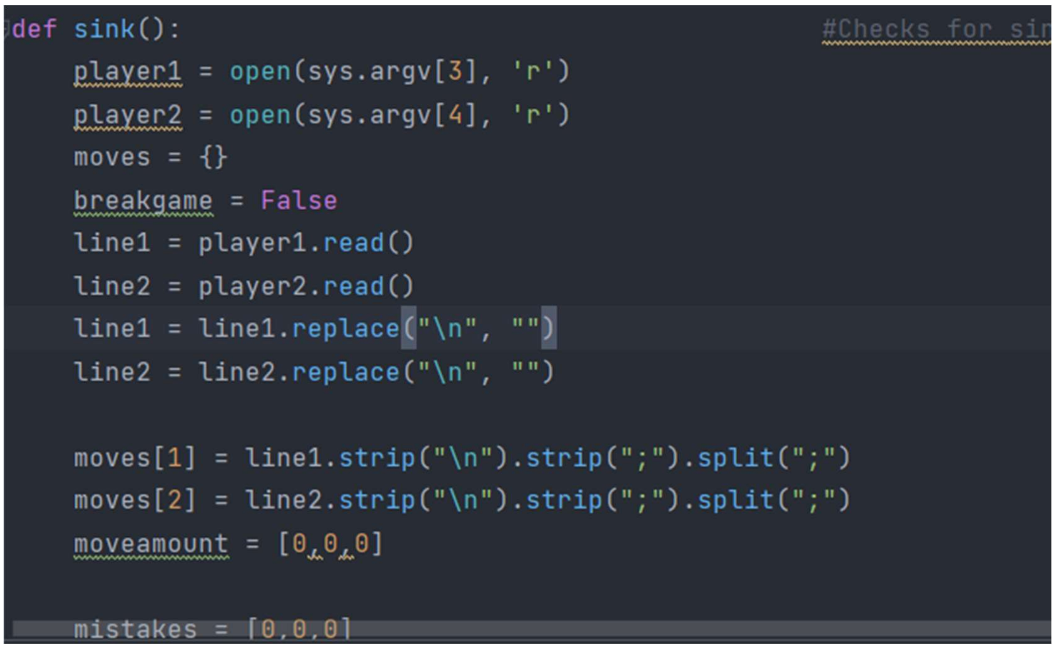


```
def gameover():
    for a in range(2):
        listship(a+1)
        i = 0
        for key in ships[a+1]:
            if ships[a+1][key] == "X":
                i += 1
            if i == len(ships[a+1]):
                over = True
                winner = round(2 - (0.3 * (a+1)))
                show(0,0, True, winner)
                breakgame = True
                return True
```

This function is for defining the game is over or not. As you can see, over turns to true when ships are sunk. Winner equation is unnecessary, but logic is this:

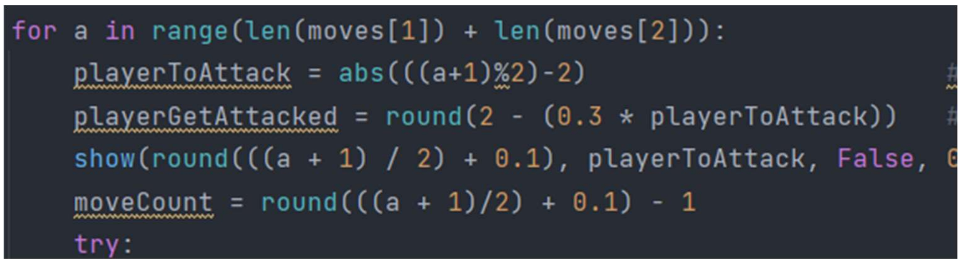
a is 0 or 1 because I said a in range (2). When a = 0; winner would be 2 because inside of the round function will be 1.7, and it rounds to 2 and this means Player 2 is the winner. When a = 1, winner would be 1 because inside of the round function would be 1.4 and it rounds to 1. Which means that Player 1 is the winner.

3.1.9 – Sink () Function:



```
def sink():                                                    #Checks for sir
    player1 = open(sys.argv[3], 'r')
    player2 = open(sys.argv[4], 'r')
    moves = {}
    breakgame = False
    line1 = player1.read()
    line2 = player2.read()
    line1 = line1.replace("\n", "")
    line2 = line2.replace("\n", "")

    moves[1] = line1.strip("\n").strip(";").split(";")
    moves[2] = line2.strip("\n").strip(";").split(";")
    moveamount = [0,0,0]

    mistakes = [0,0,0]
```

This function is for basically plays the rounds and reads the moves. Replace function works for replacing “\n” and puts nothing there. I removed \n ‘s and splatted from “;”.



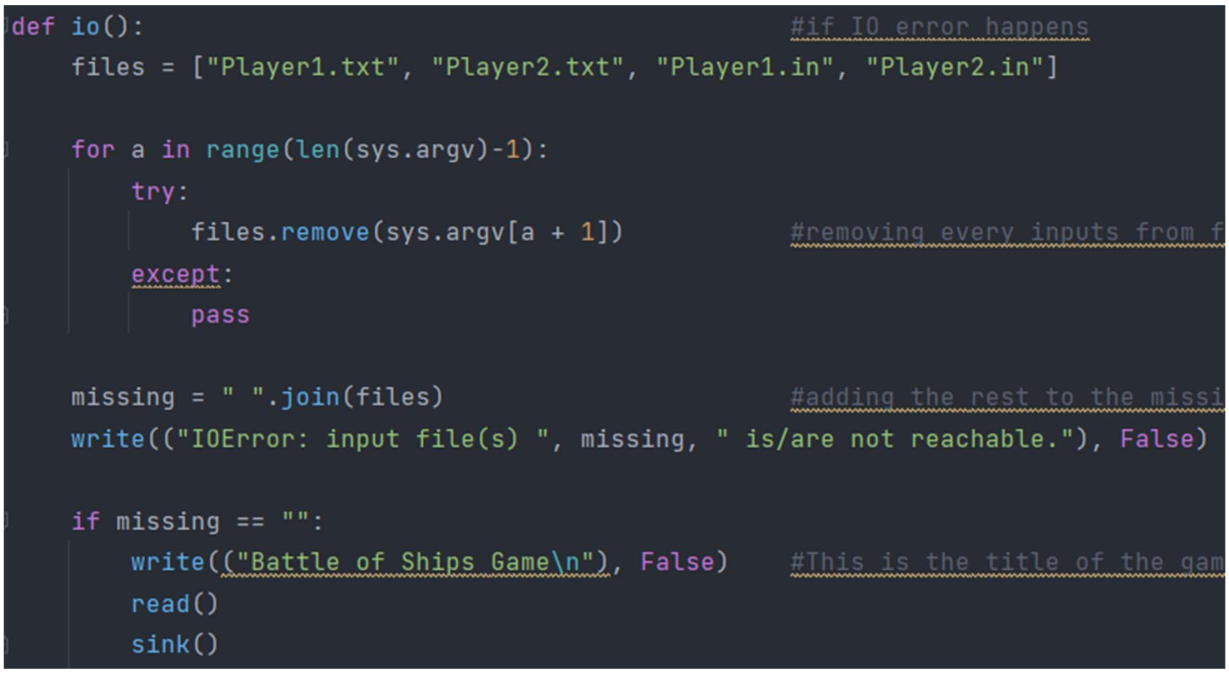
```
for a in range(len(moves[1]) + len(moves[2])):
    playerToAttack = abs(((a+1)%2)-2)                              #
    playerGetAttacked = round(2 - (0.3 * playerToAttack))          #
    show(round(((a + 1) / 2) + 0.1), playerToAttack, False, 0
    moveCount = round(((a + 1)/2) + 0.1) - 1
    try:
```

I had to mention about these calculations above because they are not making any sense at first look.

At playerToAttack part, the sum of moves1 and moves2 will give us total moves. If the total moves are odd number, then output should give us the number 1, because first player must attack at odd numbers. When total moves are odd numbers, this equation must give us 2 because second player must attack at that rounds. When you give a any numbers, you can see that this equation works fine.

The same thing is valid for the playerGetAttacked part and it is working just fine too. Again, you can see that it works correct if you try that equation.

3.1.10-IO () Function:



```
def io():                                                    #if IO error happens
    files = ["Player1.txt", "Player2.txt", "Player1.in", "Player2.in"]

    for a in range(len(sys.argv)-1):
        try:
            files.remove(sys.argv[a + 1])                    #removing every inputs from f
        except:
            pass

    missing = " ".join(files)                                #adding the rest to the missi
    write(("IOError: input file(s) ", missing, " is/are not reachable."), False)

    if missing == "":
        write(("Battle of Ships Game\n"), False)    #This is the title of the gam
        read()
        sink()
```

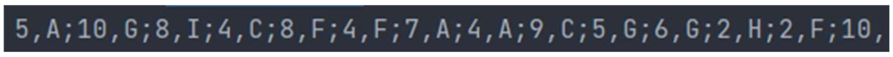
There are files that should exist in the files list. I check the list and if these files entered in sys, then function removes from files list in try part. End of the for loop, there would be missing files at the files list. I added missing files to missing empty string and then printed which files are missing. If missing is empty string end of this stages, program just prints the title of the program, calls the functions and starts working.

4 – User’s Catalogue:

1- When user wants to run program, user should not change the name of input files or try input file named differently. If user do that program would not perceive the inputs and it will give you IO error and tells you which arguments(files) are missing.

2- If you insist to use files at the names that you want, you should change the names from IO function and give names as you want.

3- Input types mustn’t change, and it has to be like this format:

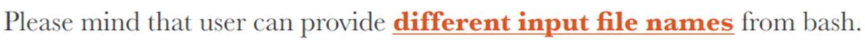


```
5,A;10,G;8,I;4,C;8,F;4,F;7,A;4,A;9,C;5,G;6,G;2,H;2,F;10,
```

4- You are not allowed change the ships position because I used optional txt file which helps position of the ships.

5- I tried to block the missing moves, but I highly recommend to enter correct inputs because there might be some cases that will crash my program or make it give error.

6- You can change inputs (where player can attack) but you are not allowed to change the format of the input like entering lower case letters.

7- 

Please mind that user can provide **different input file names** from bash.

I did not care about that because if I do that my program can not show which file is missing because file names can change.

8- Have Fun.