Hacettepe University
Department of Computer Engineering

BBM 103 Assignment 4 Report

Umut Bayındır - 2210356147

I.Table of Contents

1.Introduction……………………………………………………………………………………...3
2.Reading Functions………………………………………………………………………………….3
2.1.Reading .txt Files……………………………...……………………………………………….3
2.2.Reading .in Files……………………………………………………….……………………....4
3.Pre-Game Functions..…………………………………………………………………………5
3.1.generate()……………..………………………………………………………………………...5
3.2.board()…………..……………………………………………………………………………..6
4.play()……………………………………………………………………………………...…..7
4.1.Beautification………………………..……………………………………………………….7
4.2.Keeping the Ships' Status in Check…………………………………………………………...8
4.3.Winning……………………………………………………….……………………………….9
4.4.Final Board………………………………………………………………………………….9
5.Write………...………………………………………………………………………...……..10
6.Conclusion…………..………………………………………………………………………..10

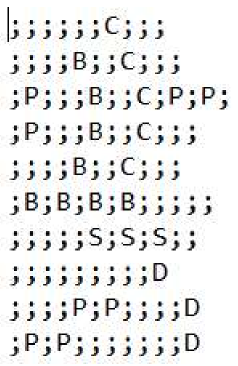
1.Introduction

In this assignment, we were tasked with reading input files and processing their data to an output file in a Battleship like setting using terminal. Battleship is a two player guessing game in which is played on ruled grids, on boards in the commercial versions, on which each player's fleet of warships are marked. The locations of the fleets are concealed from the other player. Players alternate turns calling shots at the other player's ships, and the objective of the game is to destroy the opposing player's fleet.

2.Reading Functions

We were given 4 input files, Player1.txt, Player2.txt, Player1.in and Player2.in respectively.

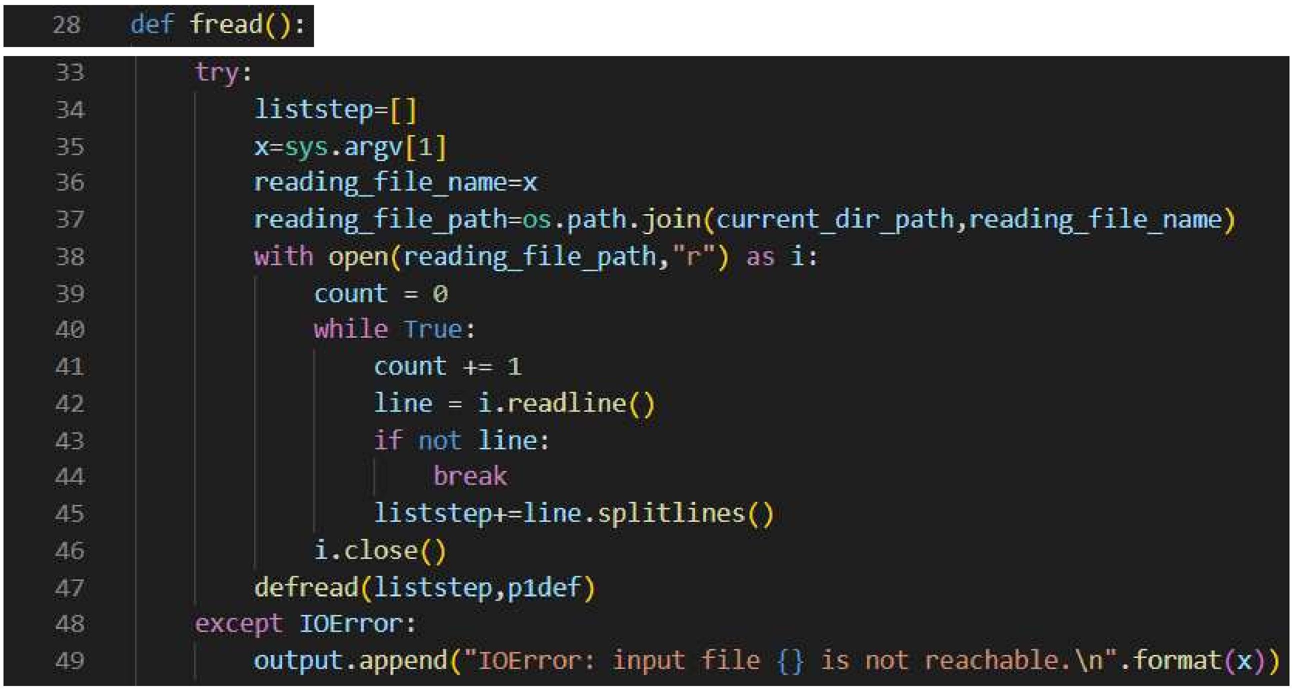
2.1.Reading .txt Files



```
;;;;;;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
```

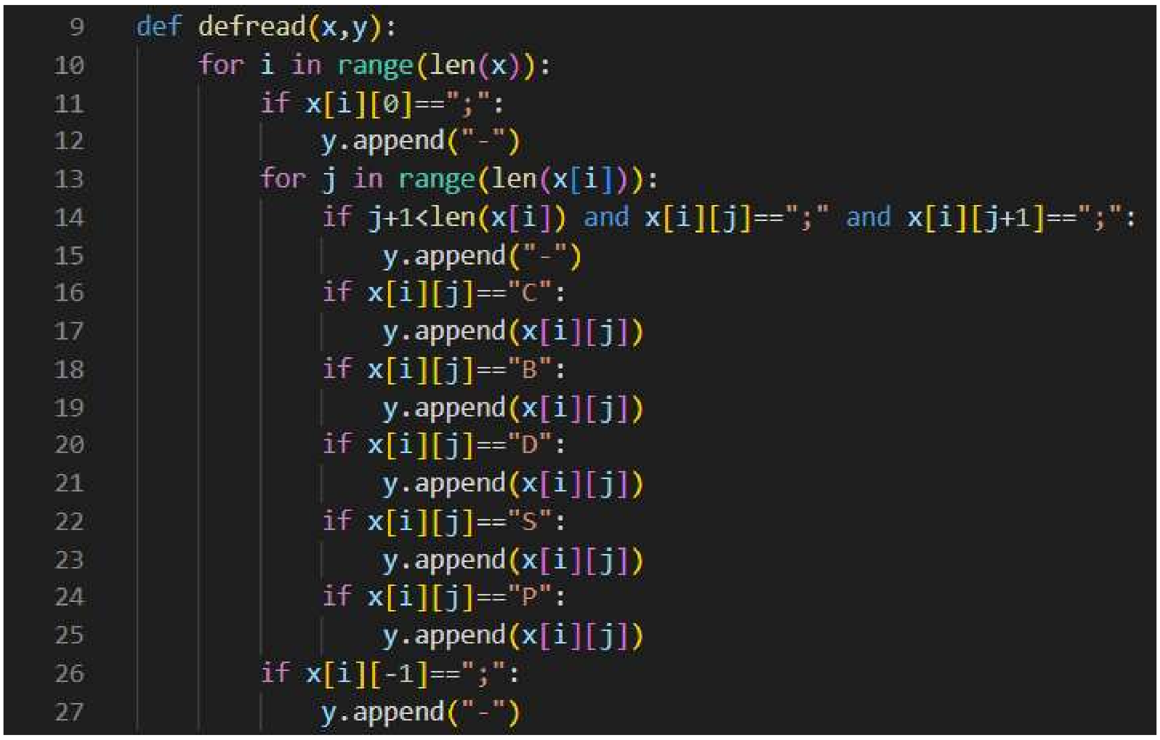
Player1.txt

.txt files designate where the players puts their ships on. Semicolons separate each tile.



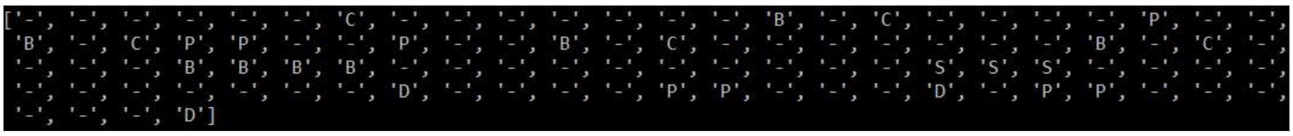
```python
def fread():
    try:
        liststep=[]
        x=sys.argv[1]
        reading_file_name=x
        reading_file_path=os.path.join(current_dir_path,reading_file_name)
        with open(reading_file_path,"r") as i:
            count = 0
            while True:
                count += 1
                line = i.readline()
                if not line:
                    break
                liststep+=line.splitlines()
            i.close()
        defread(liststep,p1def)
    except IOError:
        output.append("IOError: input file {} is not reachable.\n".format(x))
```

In this part, .txt files are split to lines if the name of the entered files are in the directory.



```python
def defread(x,y):
    for i in range(len(x)):
        if x[i][0]==";":
            y.append("-")
        for j in range(len(x[i])):
            if j+1<len(x[i]) and x[i][j]==";" and x[i][j+1]==";":
                y.append("-")
            if x[i][j]=="C":
                y.append(x[i][j])
            if x[i][j]=="B":
                y.append(x[i][j])
            if x[i][j]=="D":
                y.append(x[i][j])
            if x[i][j]=="S":
                y.append(x[i][j])
            if x[i][j]=="P":
                y.append(x[i][j])
        if x[i][-1]==";":
            y.append("-")
```

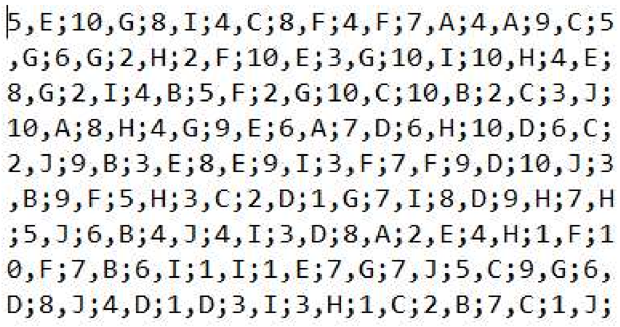
Then the lines are fed to a function named defread(). This function turns the input to a list, namely p1def for Player1.txt and p2def for Player2.txt, by creating spaces if a line starts with a semicolon, if there are two semicolons in a row or a line ends with a semicolon. It also directly adds the letters “C”,“B”,”D”,”S” or “P” if they are detected.



```
['-', '-', '-', '-', '-', '-', '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']
```

p1def in terminal

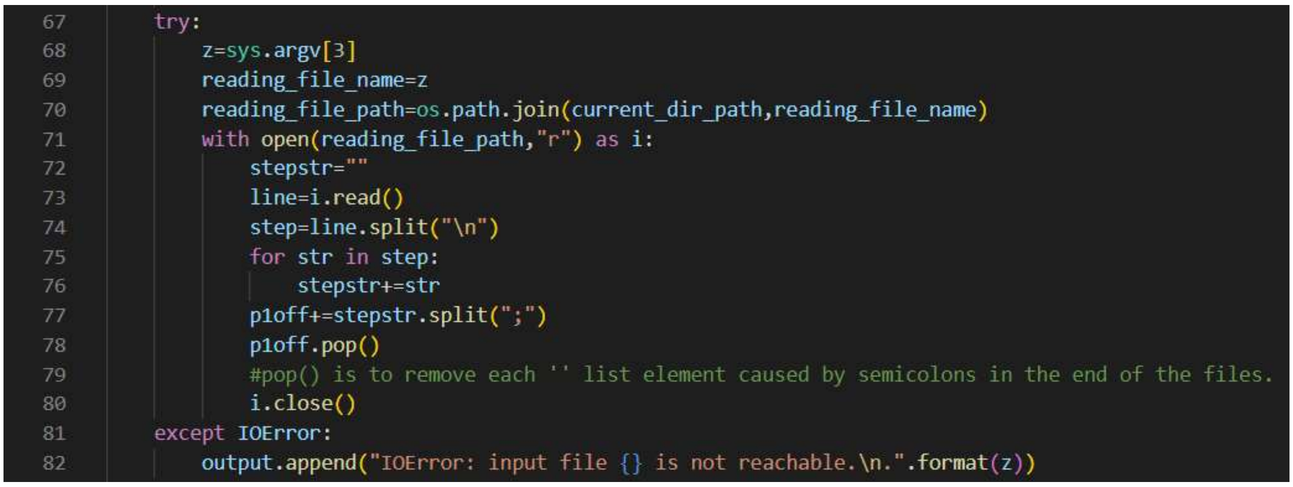
2.2.Reading .in Files



```
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,C;3,J;
10,A;8,H;4,G;9,E;6,A;7,D;6,H;10,D;6,C;
2,J;9,B;3,E;8,E;9,I;3,F;7,F;9,D;10,J;3
,B;9,F;5,H;3,C;2,D;1,G;7,I;8,D;9,H;7,H
;5,J;6,B;4,J;4,I;3,D;8,A;2,E;4,H;1,F;1
0,F;7,B;6,I;1,I;1,E;7,G;7,J;5,C;9,G;6,
D;8,J;4,D;1,D;3,I;3,H;1,C;2,B;7,C;1,J;
```

Player1.in

.in files designate where the players shoot the other player’s tiles. Semicolons separate each move.



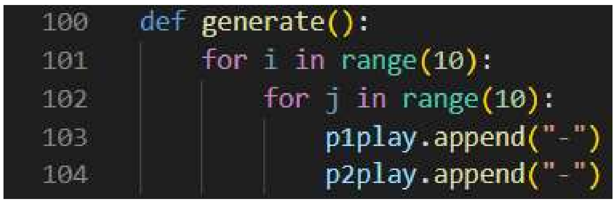
```python
    try:
        z=sys.argv[3]
        reading_file_name=z
        reading_file_path=os.path.join(current_dir_path,reading_file_name)
        with open(reading_file_path,"r") as i:
            stepstr=""
            line=i.read()
            step=line.split("\n")
            for str in step:
                stepstr+=str
            p1off+=stepstr.split(";")
            p1off.pop()
            #pop() is to remove each '' list element caused by semicolons in the end of the files.
            i.close()
    except IOError:
        output.append("IOError: input file {} is not reachable.\n".format(z))
```

Similar to .txt, .in files are split to lines if the name of the entered file is in the directory. Following this, line list is turned into string. Then .split() command splits each line to each move in p1off and p2off lists. As stated in the comment, .pop() command is used to remove the last elements,’’, caused by semicolons in the end of the files. This is done to prevent potential errors in later parts.

3.Pre-Game Functions

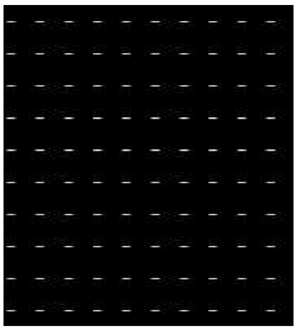
Before starting to use p1def,p2def,p1off and p2off lists to play the game, two functions are required. One to and another to make a table to make the later parts easier.

3.1.generate()



```python
def generate():
    for i in range(10):
        for j in range(10):
            p1play.append("-")
            p2play.append("-")
```

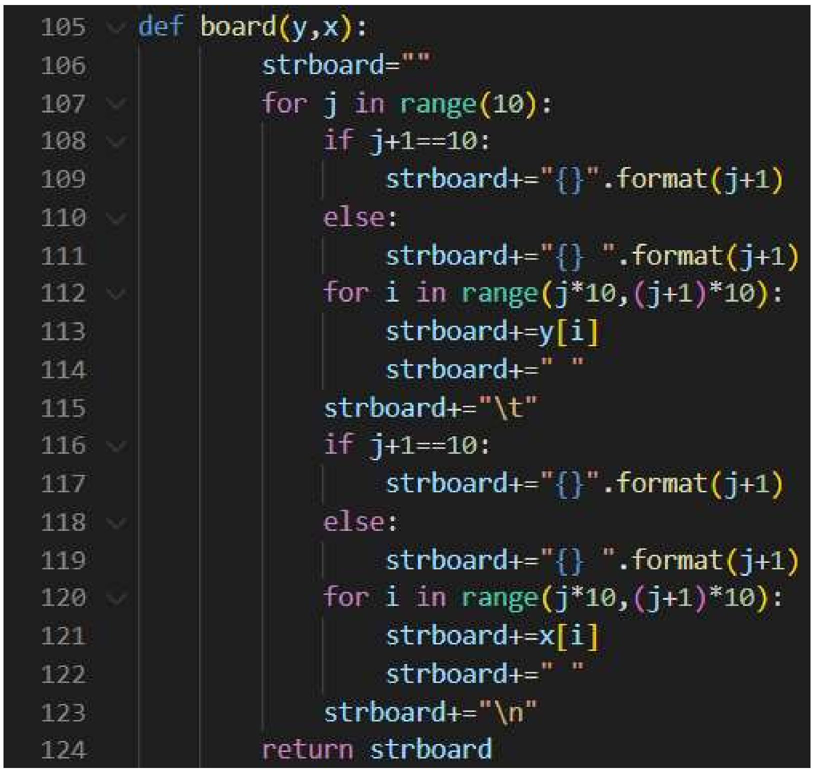
This function creates a 10x10 grid for players to see the results of their attack moves.



```
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

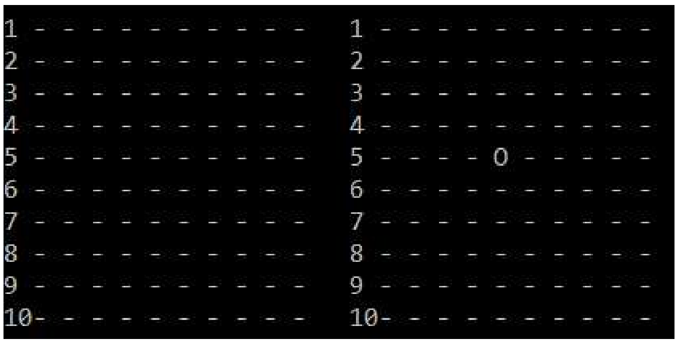
string version of p1play list

3.2.board()



```python
def board(y,x):
        strboard=""
        for j in range(10):
            if j+1==10:
                strboard+="{}".format(j+1)
            else:
                strboard+="{} ".format(j+1)
            for i in range(j*10,(j+1)*10):
                strboard+=y[i]
                strboard+=" "
            strboard+="\t"
            if j+1==10:
                strboard+="{}".format(j+1)
            else:
                strboard+="{} ".format(j+1)
            for i in range(j*10,(j+1)*10):
                strboard+=x[i]
                strboard+=" "
            strboard+="\n"
        return strboard
```

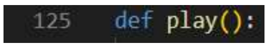
board() function has two uses. For first use, it creates two 10x10 grids representing players’ offense boards. For second use, it creates two 10x10 grids representing players’ defense boards.



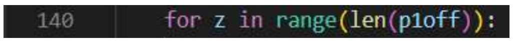
```
1 - - - - - - - - - -    1 - - - - - - - - - -
2 - - - - - - - - - -    2 - - - - - - - - - -
3 - - - - - - - - - -    3 - - - - - - - - - -
4 - - - - - - - - - -    4 - - - - - - - - - -
5 - - - - - - - - - -    5 - - - - O - - - - -
6 - - - - - - - - - -    6 - - - - - - - - - -
7 - - - - - - - - - -    7 - - - - - - - - - -
8 - - - - - - - - - -    8 - - - - - - - - - -
9 - - - - - - - - - -    9 - - - - - - - - - -
10- - - - - - - - - -    10- - - - - - - - - -
```

strboard in use

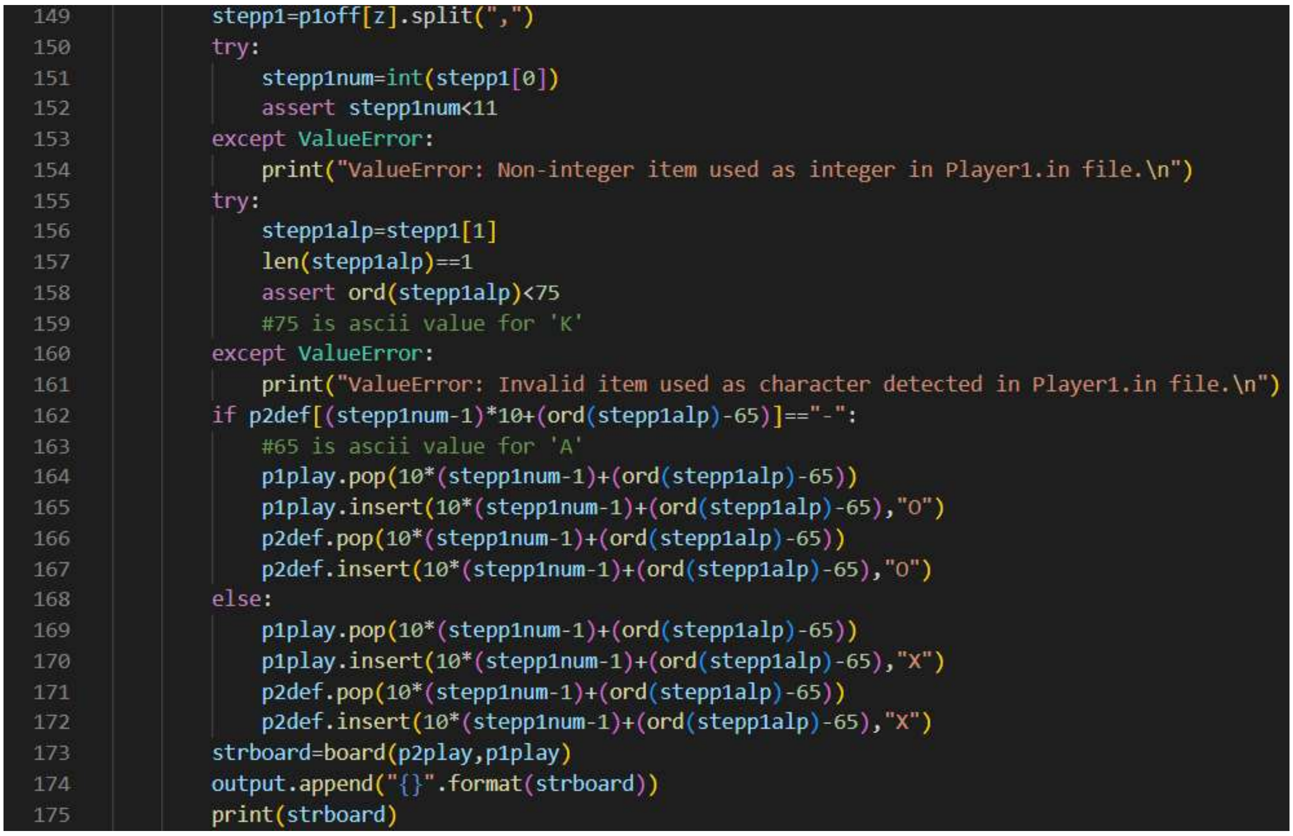
4.play()



```python
def play():
```



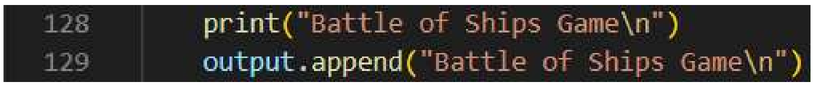
```python
    for z in range(len(p1off)):
```



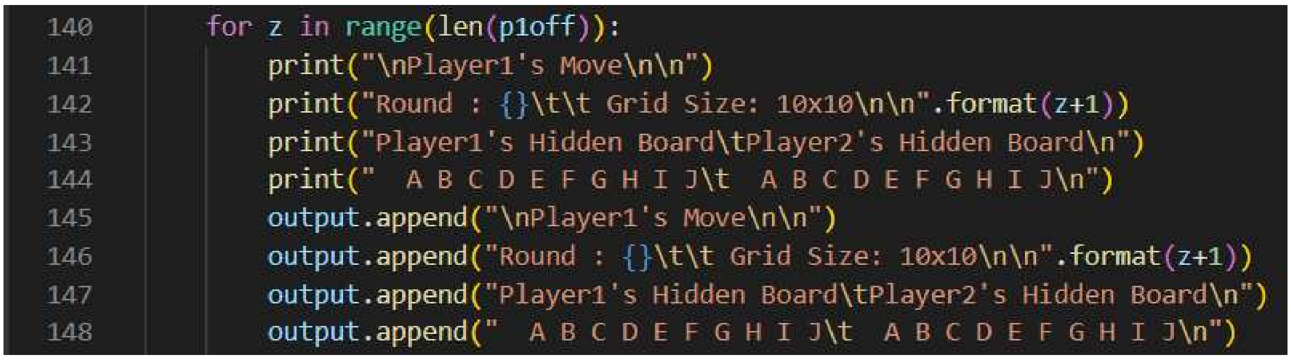
```python
        stepp1=p1off[z].split(",")
        try:
            stepp1num=int(stepp1[0])
            assert stepp1num<11
        except ValueError:
            print("ValueError: Non-integer item used as integer in Player1.in file.\n")
        try:
            stepp1alp=stepp1[1]
            len(stepp1alp)==1
            assert ord(stepp1alp)<75
            #75 is ascii value for 'K'
        except ValueError:
            print("ValueError: Invalid item used as character detected in Player1.in file.\n")
        if p2def[(stepp1num-1)*10+(ord(stepp1alp)-65)]=="-":
            #65 is ascii value for 'A'
            p1play.pop(10*(stepp1num-1)+(ord(stepp1alp)-65))
            p1play.insert(10*(stepp1num-1)+(ord(stepp1alp)-65),"O")
            p2def.pop(10*(stepp1num-1)+(ord(stepp1alp)-65))
            p2def.insert(10*(stepp1num-1)+(ord(stepp1alp)-65),"O")
        else:
            p1play.pop(10*(stepp1num-1)+(ord(stepp1alp)-65))
            p1play.insert(10*(stepp1num-1)+(ord(stepp1alp)-65),"X")
            p2def.pop(10*(stepp1num-1)+(ord(stepp1alp)-65))
            p2def.insert(10*(stepp1num-1)+(ord(stepp1alp)-65),"X")
        strboard=board(p2play,p1play)
        output.append("{}".format(strboard))
        print(strboard)
```

In this part, we will look at play(), the most important function in the code. This function uses the moves in p1off and p2off on p2def and p1def respectively, as well as on p1play and p2play respectively. First of all, this function strips the round's move to two parts, number and letter. These values are used to find the location of the desired tile by multiplying the number by 10 for row. For column, adding the value of the letter by finding the ascii value of the letter and then subtracting it by 65, A for 0, B for 1 and so on. Every time a change happens on the boards, it's shown in print in terminal and on output file on each player's turn.

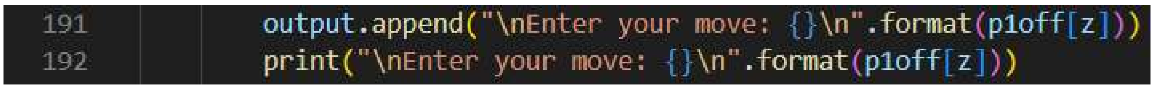
4.1.Beautification



```python
    print("Battle of Ships Game\n")
    output.append("Battle of Ships Game\n")
```

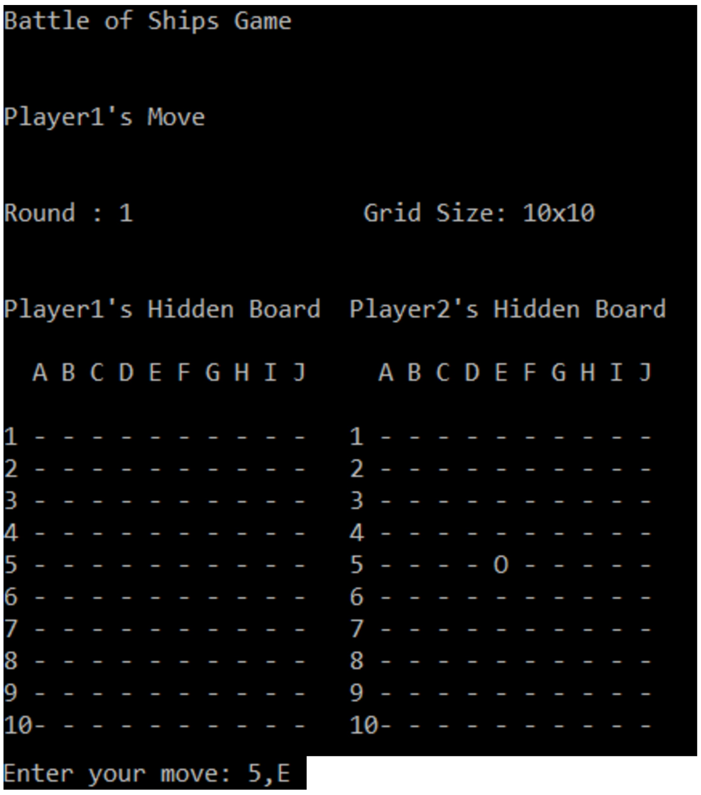


```python
    for z in range(len(p1off)):
        print("\nPlayer1's Move\n\n")
        print("Round : {}\t\t Grid Size: 10x10\n\n".format(z+1))
        print("Player1's Hidden Board\tPlayer2's Hidden Board\n")
        print("  A B C D E F G H I J\t  A B C D E F G H I J\n")
        output.append("\nPlayer1's Move\n\n")
        output.append("Round : {}\t\t Grid Size: 10x10\n\n".format(z+1))
        output.append("Player1's Hidden Board\tPlayer2's Hidden Board\n")
        output.append("  A B C D E F G H I J\t  A B C D E F G H I J\n")
```



```
output.append("\nEnter your move: {}\n".format(p1off[z]))
print("\nEnter your move: {}\n".format(p1off[z]))
```

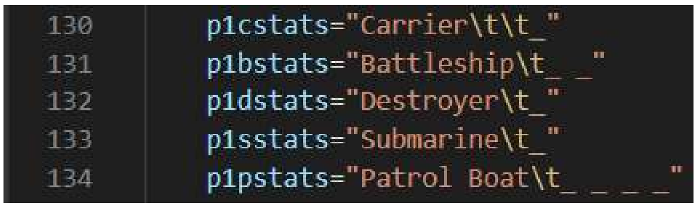
There are also some extra details done to add to the overall view of the rounds, showing the title, notifying whose turn it is, whose board is whose and showing the columns with letters.



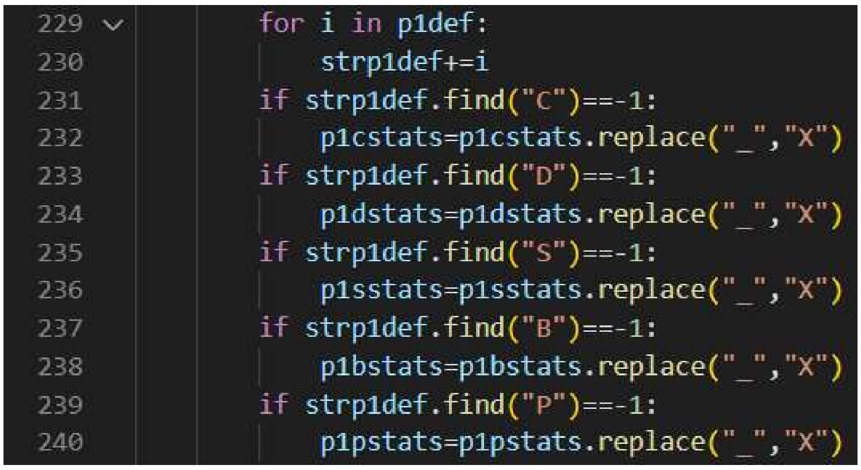
```
Battle of Ships Game


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 - - - - O - - - - -
6 - - - - - - - - - -   6 - - - - - - - - - -
7 - - - - - - - - - -   7 - - - - - - - - - -
8 - - - - - - - - - -   8 - - - - - - - - - -
9 - - - - - - - - - -   9 - - - - - - - - - -
10- - - - - - - - - -   10- - - - - - - - - -

Enter your move: 5,E
```

player 1’s first move on terminal

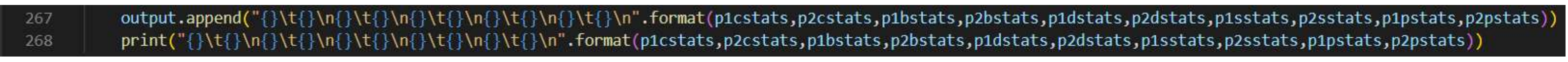
4.2.Keeping the Ships’ Status in Check



```
p1cstats="Carrier\t\t_"
p1bstats="Battleship\t_ _"
p1dstats="Destroyer\t_"
p1sstats="Submarine\t_"
p1pstats="Patrol Boat\t_ _ _ _"
```

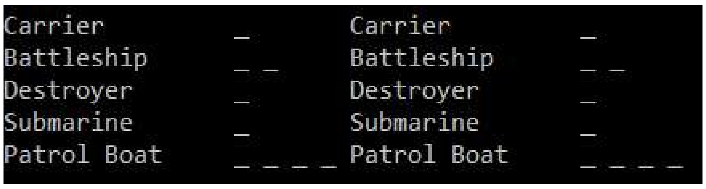


```
for i in p1def:
    strp1def+=i
if strp1def.find("C")==-1:
    p1cstats=p1cstats.replace("_","X")
if strp1def.find("D")==-1:
    p1dstats=p1dstats.replace("_","X")
if strp1def.find("S")==-1:
    p1sstats=p1sstats.replace("_","X")
if strp1def.find("B")==-1:
    p1bstats=p1bstats.replace("_","X")
if strp1def.find("P")==-1:
    p1pstats=p1pstats.replace("_","X")
```



```python
    output.append("{}\t{}\n{}\t{}\n{}\t{}\n{}\t{}\n{}\t{}\n".format(p1cstats,p2cstats,p1bstats,p2bstats,p1dstats,p2dstats,p1sstats,p2sstats,p1pstats,p2pstats))
    print("{}\t{}\n{}\t{}\n{}\t{}\n{}\t{}\n{}\t{}\n".format(p1cstats,p2cstats,p1bstats,p2bstats,p1dstats,p2dstats,p1sstats,p2sstats,p1pstats,p2pstats))
```

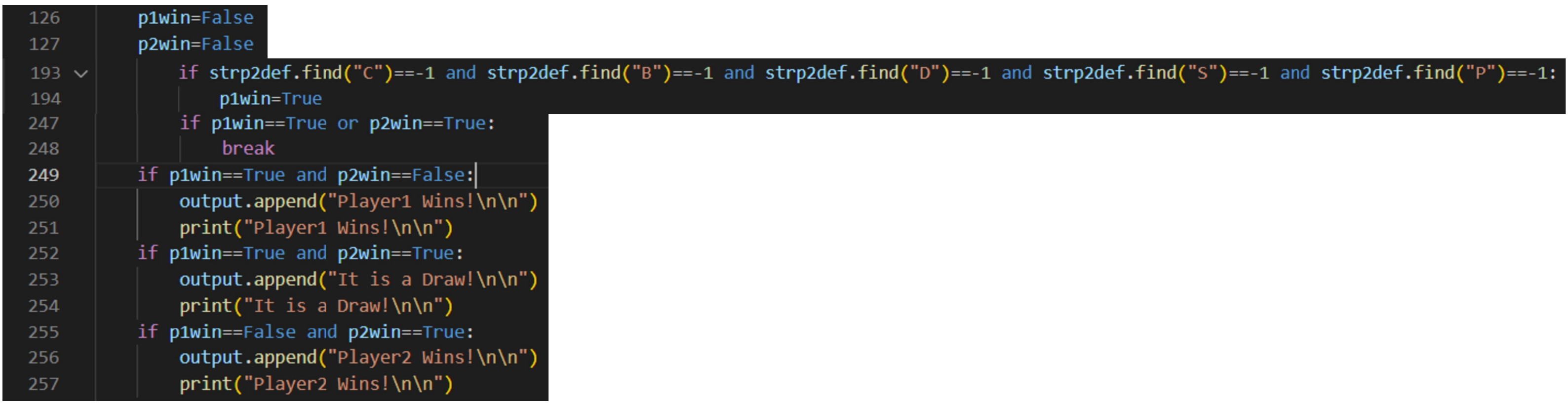
Another important detail during a round is players’ ships status. I have designed it in a way so the ships will show cross next to them if a tile with a ship’s respective letter cannot be found. I originally had a plan to show each ship for the battleship and patrol boat classes, but I couldn’t come up a way to code it on time.



```
Carrier             _        Carrier             _
Battleship          _ _      Battleship          _ _
Destroyer           _        Destroyer           _
Submarine           _        Submarine           _
Patrol Boat         _ _ _ _  Patrol Boat         _ _ _ _
```

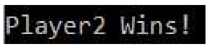
Ships’ status on terminal

4.3.Winning



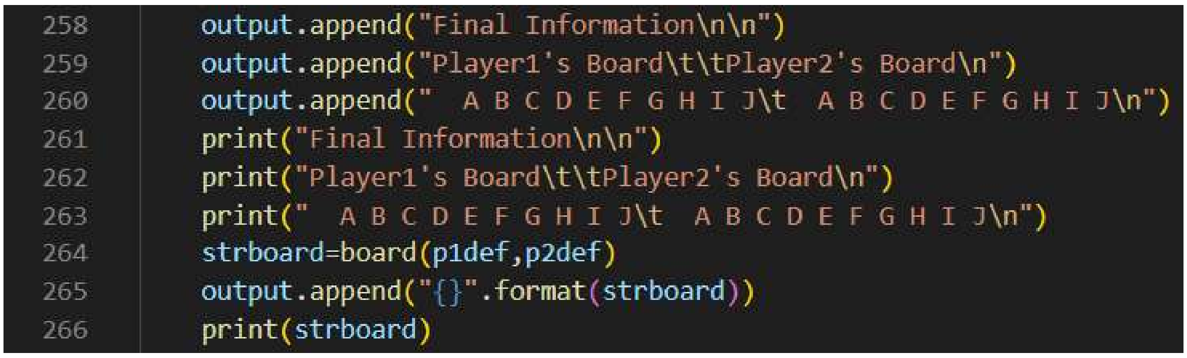
```python
    p1win=False
    p2win=False
        if strp2def.find("C")==-1 and strp2def.find("B")==-1 and strp2def.find("D")==-1 and strp2def.find("S")==-1 and strp2def.find("P")==-1:
            p1win=True
        if p1win==True or p2win==True:
            break
    if p1win==True and p2win==False:
        output.append("Player1 Wins!\n\n")
        print("Player1 Wins!\n\n")
    if p1win==True and p2win==True:
        output.append("It is a Draw!\n\n")
        print("It is a Draw!\n\n")
    if p1win==False and p2win==True:
        output.append("Player2 Wins!\n\n")
        print("Player2 Wins!\n\n")
```

If a player manages to take out all the ships of the player, they win. If they manage to defeat each other on the same turn, then it’s a draw.



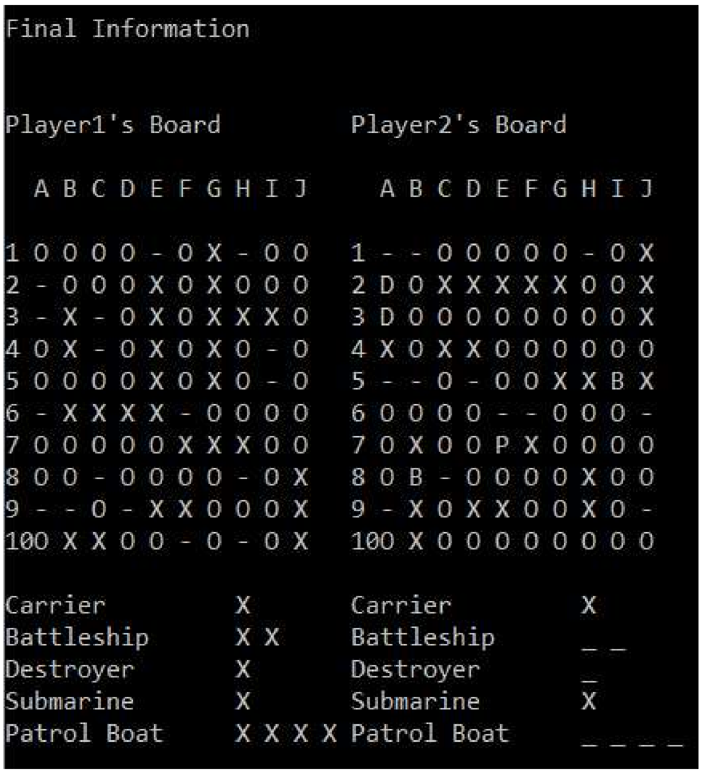
```
Player2 Wins!
```

4.4.Final Board



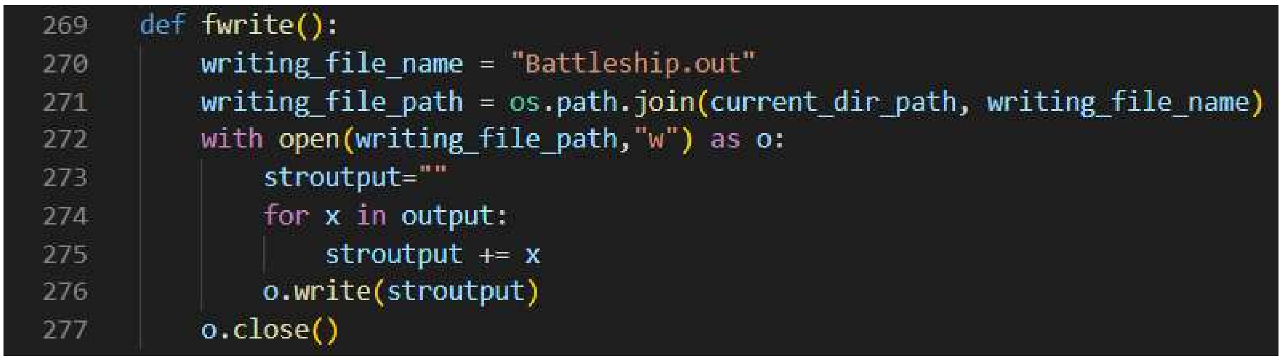
```python
    output.append("Final Information\n\n")
    output.append("Player1's Board\t\tPlayer2's Board\n")
    output.append("  A B C D E F G H I J\t  A B C D E F G H I J\n")
    print("Final Information\n\n")
    print("Player1's Board\t\tPlayer2's Board\n")
    print("  A B C D E F G H I J\t  A B C D E F G H I J\n")
    strboard=board(p1def,p2def)
    output.append("{}".format(strboard))
    print(strboard)
```

When the match ends, two boards, this time the actual defense boards and not offense boards, will be shown, along with ships’ status.



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

5.Write



```python
def fwrite():
    writing_file_name = "Battleship.out"
    writing_file_path = os.path.join(current_dir_path, writing_file_name)
    with open(writing_file_path,"w") as o:
        stroutput=""
        for x in output:
            stroutput += x
        o.write(stroutput)
    o.close()
```

While the rounds are being printed, records of the print will also be saved to an output file titled Battleship.out.

6.Conclusion

This assignment showcases python can be used to make complex tasks, such as simulating a game such as the Battleship. This assignment helped me in practicing the methods I’ve previously learned.