Hacettepe University Computer Engineering Department

BBM103 Introduction to Programming Lab 1

Assignment 4: Battle of Ships

Efe AYDINALP

Programmers Catalogue

This code written in 4 days. Design of the code has three part.

First read informations of ships location from player(s) txt file, convert them to coordinates and add them to lists for making them more readable meanwhile make easy to printing process.In the same time in this way we can check if a ship still floating or not. Then the code creates hidden boards which ones actually will become game board.

There is files whic ones names are “OptionalPlayer1.txt” and “OptionalPlayer2.txt”. Program uses this files to understand some spesific ships coordinates (Battleship and Patroal Boat).

Every function in this part and Optional Player files will explained in Design part detailed.

After this operations the code moves on to the second part. In this part the code read informations of moves from player.in file for each player. Convert them to matrix versions of them like if player wants to shoot (1,B) , it’s matrix version equals to (0,1). So we can use move input with lists which ones contains ship coordinates.

Every function in this part will explained in Design part detailed.

Finally after we read ship locations and move inputs from player, there is a control()

function before game() function. Control() function does just check if someone won by looking if ships of one of the players did sink. Then game() function calculates which player’s turn, calculates round, make moves, print hidden boards by calling function, check if someone won with control() function. Game() runs until someone win or one of the players move’s finish etc.

Game() and control() functions explained in Design part detailed.

Sample Run:

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

Program will crush if you do not run it in this order from terminal.

How To Test Errors :

It’s replayable game. If you change input and txt files it may change result.

You can test the program with changing moves in input file like

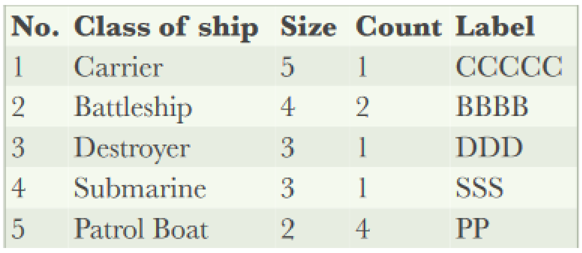
(11,A;11,11;A,A... etc.)

About how write input and txt files please check User Catalogue part.

User Catalogue

Defining Ship Locations and Txt Files

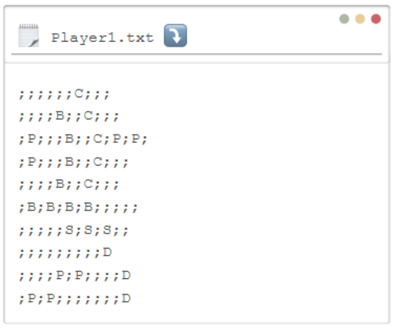
Board size is 10x10. Vertical vector areas showing with numbers (1,2,3,4,5,6,7,8,9,10). Horizontal vector areas showing with letters (A,B,C,D,E,F,G,H,I,J). Ship names are “Carrier”, “Battleship”, “Destroyer”, “Submarine”, “Patrol Boat”.



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

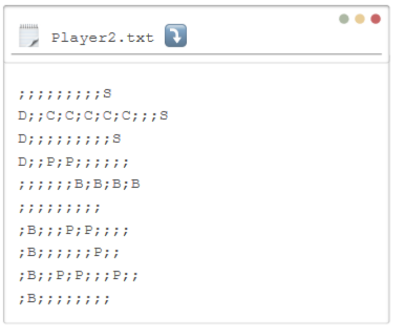
Create a txt file to show ships location or each player. To symbolize ships, you can use first letters of ships. To symbolize empty areas use “ ; “.

Every “ ; ” means empty areas and first letter of ship means that area is filled with that type of ship. You should write “ ; “ between two ship symbol as well, program needs that to understand there is two ship side by side. Program can seperate semicolons for empty places and semicolons between two ship symbol so do not worry about it.



Player1.txt

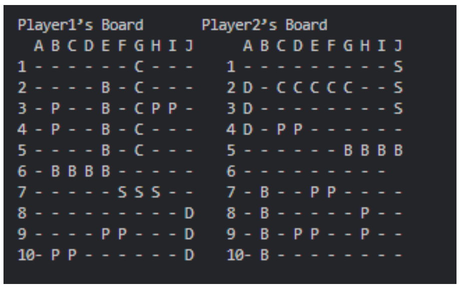
```
;;;;;;C;;;
;;;;B;;C;;;
;P;;;B;;C;P;P;
;P;;;B;;C;;;
;;;;B;;C;;;
;B;B;B;B;;;;;
;;;;;S;S;S;;
;;;;;;;;;D
;;;;P;P;;;;D
;P;P;;;;;;;D
```



Player2.txt

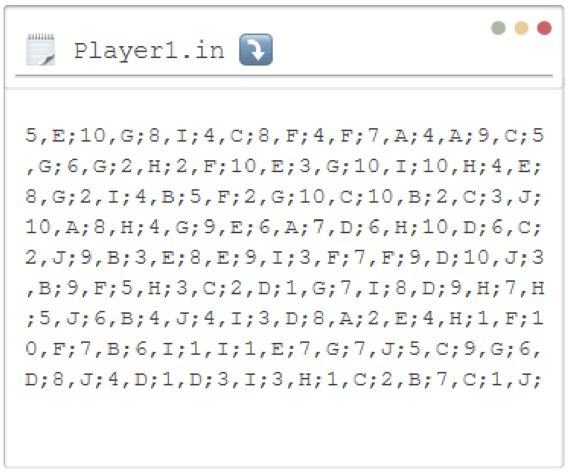
```
;;;;;;;;;S
D;;C;C;C;C;C;;;S
D;;;;;;;;;S
D;;P;P;;;;;;
;;;;;;B;B;B;B
;;;;;;;;;
;B;;;P;P;;;;
;B;;;;;;P;;
;B;;P;P;;;P;;
;B;;;;;;;;
```

In this example, Ship locations are going to look like that.

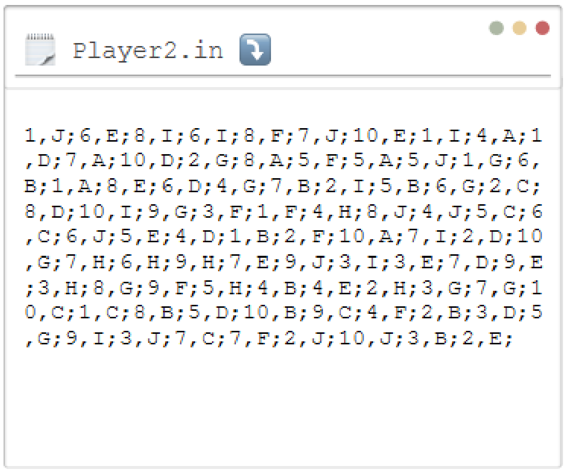


```
Player1's Board       Player2's Board
  A B C D E F G H I J     A B C D E F G H I J
1 - - - - - - C - - -   1 - - - - - - - - - S
2 - - - - B - C - - -   2 D - C C C C C - - S
3 - P - - B - C P P -   3 D - - - - - - - - S
4 - P - - B - C - - -   4 D - P P - - - - - -
5 - - - - B - C - - -   5 - - - - - - B B B B
6 - B B B B - - - - -   6 - - - - - - - - -
7 - - - - - S S S - -   7 - B - - P P - - - -
8 - - - - - - - - - D   8 - B - - - - - P - -
9 - - - - P P - - - D   9 - B - P P - - P - -
10- P P - - - - - - D   10- B - - - - - - - -
```

Defining Moves



```
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;
```



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

Create input file to moves coordinates for each player. Each moves seperated by semicolon.

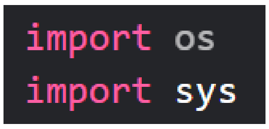
For example 5,E; means shoot the coordinate (5,E)

Design

Program has three main part. Read input files (moves) , Read txt files (ship locations) and game operation (game() function and control() function).

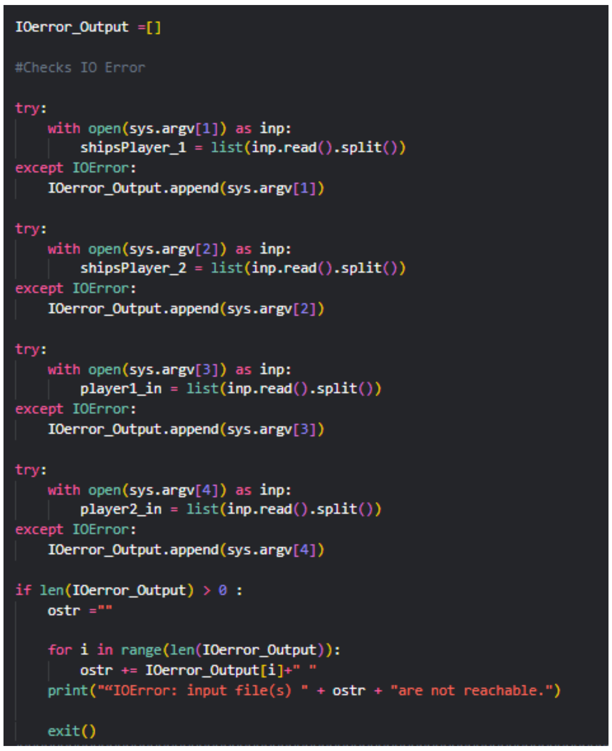
First Part – Reading Txt Files and Organize Them

Libraries:



```
import os
import sys
```

Check IO Error:



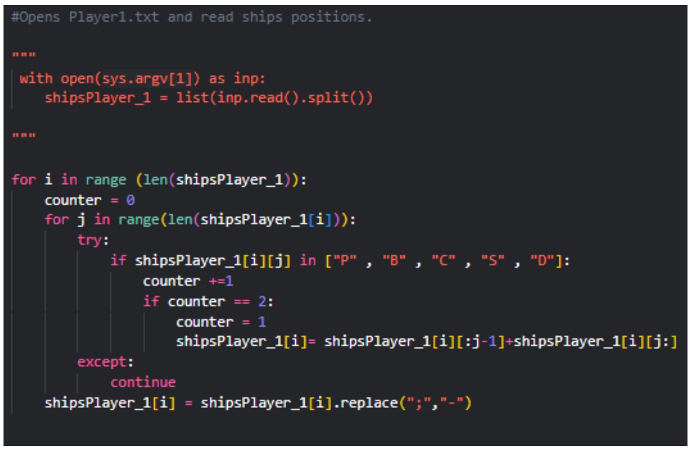
```
IOerror_Output =[]

#Checks IO Error

try:
    with open(sys.argv[1]) as inp:
        shipsPlayer_1 = list(inp.read().split())
except IOError:
    IOerror_Output.append(sys.argv[1])

try:
    with open(sys.argv[2]) as inp:
        shipsPlayer_2 = list(inp.read().split())
except IOError:
    IOerror_Output.append(sys.argv[2])

try:
    with open(sys.argv[3]) as inp:
        player1_in = list(inp.read().split())
except IOError:
    IOerror_Output.append(sys.argv[3])

try:
    with open(sys.argv[4]) as inp:
        player2_in = list(inp.read().split())
except IOError:
    IOerror_Output.append(sys.argv[4])

if len(IOerror_Output) > 0 :
    ostr =""

    for i in range(len(IOerror_Output)):
        ostr += IOerror_Output[i]+" "
    print("“IOError: input file(s) " + ostr + "are not reachable.")

    exit()
```

Sample Run:

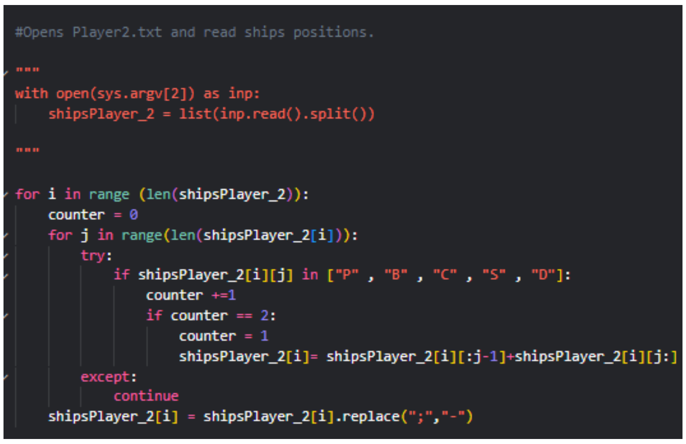
>python3 Assignment4.py “Player1.txt” “Player2.txt” “Player1.in” “Player2.in”

IOError: A problem can occur if the input files are not reachable at the specified file path (if files do not exist, or if their names are misspelled, etc.). Code is executed while one or more of the input files (“Player1.txt”, “Player2.txt”, “Player1.in”, and “Player2.in”) do not exist, Code will print the warning prompt to the output file “IOError: input file(s) Player1.txt is/are not reachable.” or “IOError: input file(s) Player1.txt, Player2.txt, Player1.in, Player2.in is/are not reachable.”.

Read Ship Positions:



```
#Opens Player1.txt and read ships positions.

"""
with open(sys.argv[1]) as inp:
    shipsPlayer_1 = list(inp.read().split())

"""

for i in range (len(shipsPlayer_1)):
    counter = 0
    for j in range(len(shipsPlayer_1[i])):
        try:
            if shipsPlayer_1[i][j] in ["P" , "B" , "C" , "S" , "D"]:
                counter +=1
                if counter == 2:
                    counter = 1
                    shipsPlayer_1[i]= shipsPlayer_1[i][:j-1]+shipsPlayer_1[i][j:]
        except:
            continue
    shipsPlayer_1[i] = shipsPlayer_1[i].replace(";","-")
```

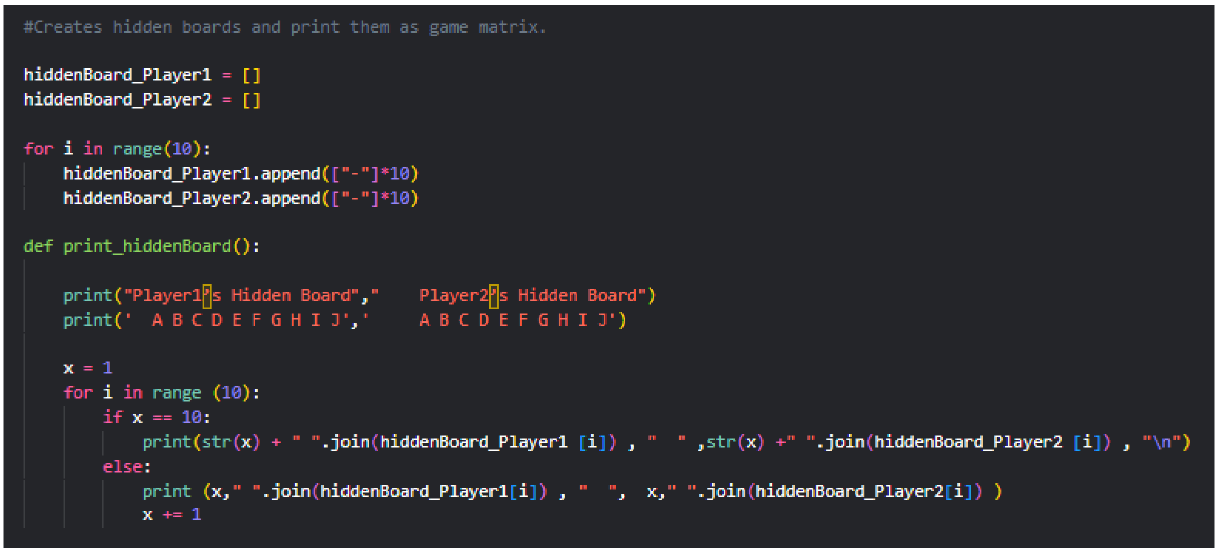


```
#Opens Player2.txt and read ships positions.

"""
with open(sys.argv[2]) as inp:
    shipsPlayer_2 = list(inp.read().split())

"""

for i in range (len(shipsPlayer_2)):
    counter = 0
    for j in range(len(shipsPlayer_2[i])):
        try:
            if shipsPlayer_2[i][j] in ["P" , "B" , "C" , "S" , "D"]:
                counter +=1
                if counter == 2:
                    counter = 1
                    shipsPlayer_2[i]= shipsPlayer_2[i][:j-1]+shipsPlayer_2[i][j:]
        except:
            continue
    shipsPlayer_2[i] = shipsPlayer_2[i].replace(";","-")
```

Function does pull every ship symbol from txt file and make every line a list then gathering all line-lists together for each player in for loop.

While doing that if there is a semicolon function control if it is between two ship symbol or not. If it is not then it is a empty area. But if it is between two ship symbol then function ignore that semicolone.

Hidden Boards:

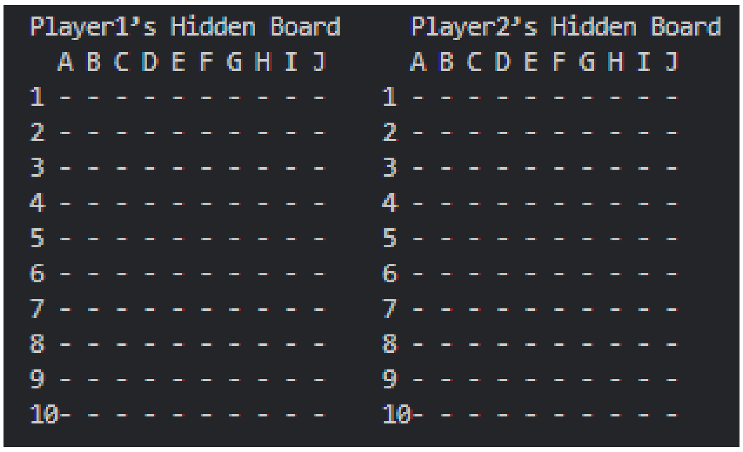


```python
#Creates hidden boards and print them as game matrix.

hiddenBoard_Player1 = []
hiddenBoard_Player2 = []

for i in range(10):
    hiddenBoard_Player1.append(["-"]*10)
    hiddenBoard_Player2.append(["-"]*10)

def print_hiddenBoard():

    print("Player1’s Hidden Board","    Player2’s Hidden Board")
    print('  A B C D E F G H I J','     A B C D E F G H I J')

    x = 1
    for i in range (10):
        if x == 10:
            print(str(x) + " ".join(hiddenBoard_Player1 [i]) , "  " ,str(x) +" ".join(hiddenBoard_Player2 [i]) , "\n")
        else:
            print (x," ".join(hiddenBoard_Player1[i]) , "  ",  x," ".join(hiddenBoard_Player2[i]) )
            x += 1
```

In this part, it first creates two seperate hidden board then the function print_hiddenboard()

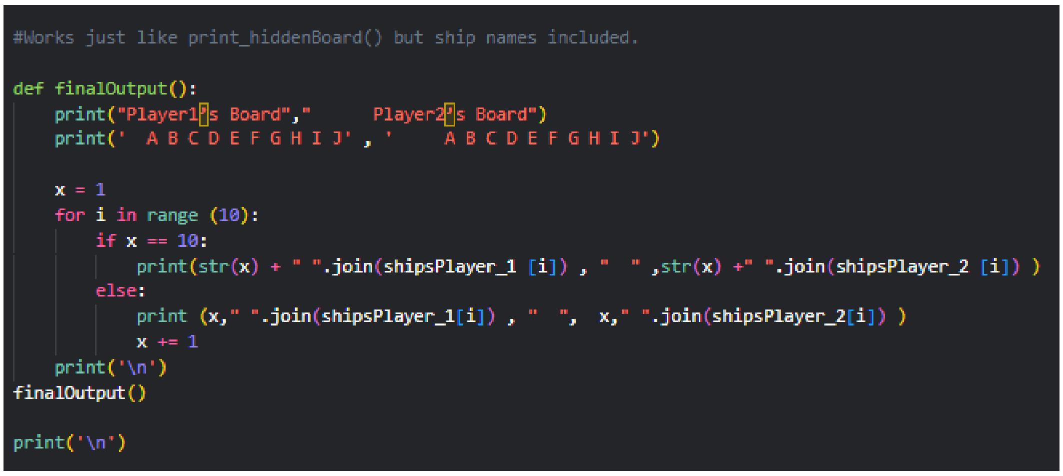
Prints “Player1’s Hidden Board”,” Player2’ Hidden Board” and horizontal vector symbols (A,B,C,D,E,F,G,H,I,J). After that function print hidden boards line by line.

It looks like that:



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

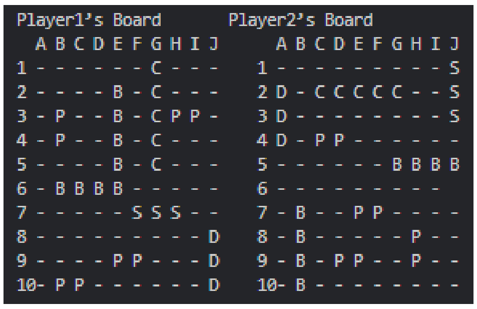
Printing Final Output – Printing ships in game board



```python
#Works just like print_hiddenBoard() but ship names included.

def finalOutput():
    print("Player1’s Board","      Player2’s Board")
    print('  A B C D E F G H I J' , '     A B C D E F G H I J')

    x = 1
    for i in range (10):
        if x == 10:
            print(str(x) + " ".join(shipsPlayer_1 [i]) , "  " ,str(x) +" ".join(shipsPlayer_2 [i]) )
        else:
            print (x," ".join(shipsPlayer_1[i]) , "  ",  x," ".join(shipsPlayer_2[i]) )
            x += 1
    print('\n')
finalOutput()

print('\n')
```

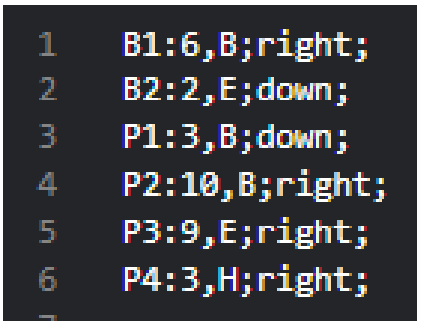
Just like printhiddenboard() function but this time ship symbols included.

Output will be like that:



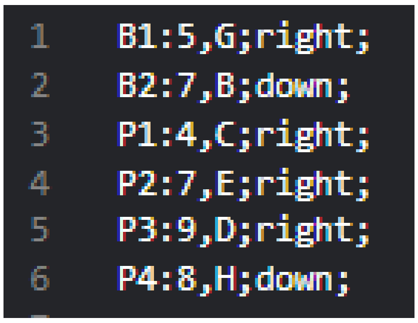
```
Player1’s Board       Player2’s Board
  A B C D E F G H I J      A B C D E F G H I J
1 - - - - - - C - - -    1 - - - - - - - - - S
2 - - - - B - C - - -    2 D - C C C C C - - S
3 - P - - B - C P P -    3 D - - - - - - - - S
4 - P - - B - C - - -    4 D - P P - - - - - -
5 - - - - B - C - - -    5 - - - - - - B B B B
6 - B B B B - - - - -    6 - - - - - - - - -
7 - - - - - S S S - -    7 - B - - P P - - - -
8 - - - - - - - - - D    8 - B - - - - - P - -
9 - - - - P P - - - D    9 - B - P P - - P - -
10- P P - - - - - - D    10- B - - - - - - - -
```

Optional Player Files:



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

OptionalPlayer1.txt



```
B1:5,G;right;
B2:7,B;down;
P1:4,C;right;
P2:7,E;right;
P3:9,D;right;
P4:8,H;down;
```

OptionalPlayer2.txt

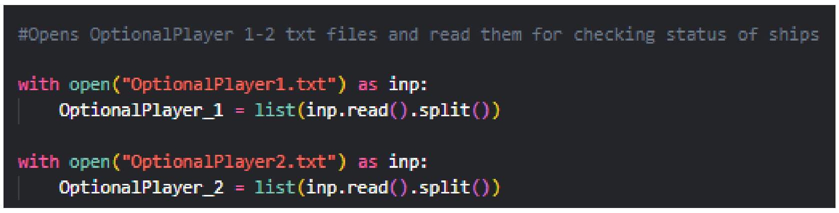
For example:

B1:6,B;right; means

Battleship 1 starts from 6,B and goes right for length of ship

It explained in detailed in Find Exact Coordinates of Ships In Optional Player Files.

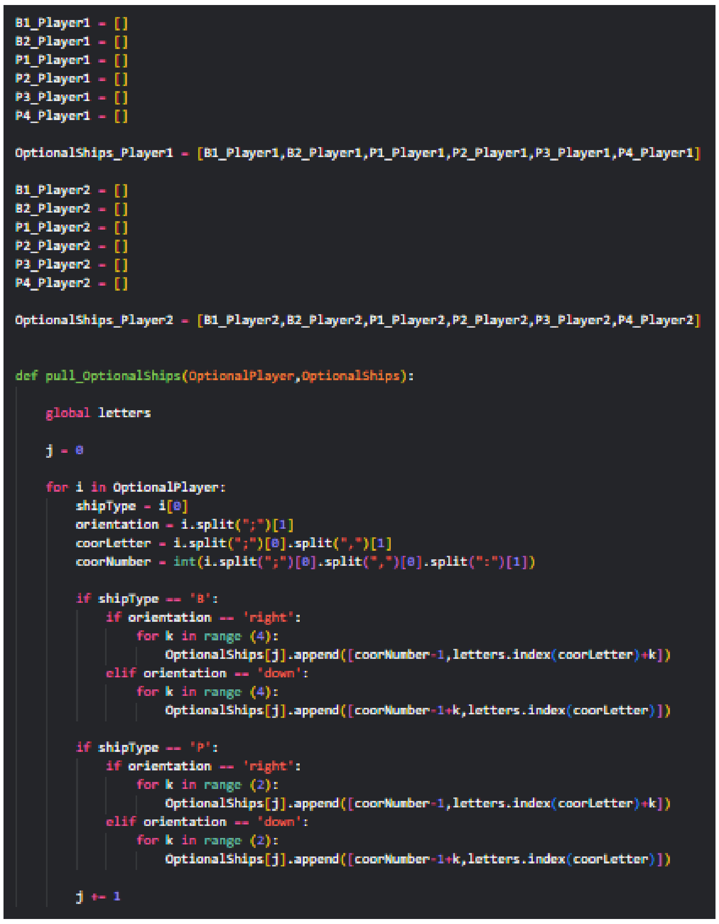
Theese files should be in folder which Program.py included. Their missions are spesify some ships (Battleship and Patroal Boat) exact coordinates. We use them because if there is more than one ship for a ship type it may broke the code. (If we do not know exact coordinate of type of ships which ones have more than one ship then we can not control which ships float, which ships sinked.)



```python
#Opens OptionalPlayer 1-2 txt files and read them for checking status of ships

with open("OptionalPlayer1.txt") as inp:
    OptionalPlayer_1 = list(inp.read().split())

with open("OptionalPlayer2.txt") as inp:
    OptionalPlayer_2 = list(inp.read().split())
```

This part reads optional player txt files.

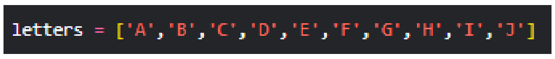
Find Exact Coordinates of Ships In Optional Player Files:



```python
B1_Player1 = []
B2_Player1 = []
P1_Player1 = []
P2_Player1 = []
P3_Player1 = []
P4_Player1 = []

OptionalShips_Player1 = [B1_Player1,B2_Player1,P1_Player1,P2_Player1,P3_Player1,P4_Player1]

B1_Player2 = []
B2_Player2 = []
P1_Player2 = []
P2_Player2 = []
P3_Player2 = []
P4_Player2 = []

OptionalShips_Player2 = [B1_Player2,B2_Player2,P1_Player2,P2_Player2,P3_Player2,P4_Player2]


def pull_OptionalShips(OptionalPlayer,OptionalShips):

    global letters

    j = 0

    for i in OptionalPlayer:
        shipType = i[0]
        orientation = i.split(";")[1]
        coorLetter = i.split(";")[0].split(",")[1]
        coorNumber = int(i.split(";")[0].split(",")[0].split(":")[1])

        if shipType == 'B':
            if orientation == 'right':
                for k in range (4):
                    OptionalShips[j].append([coorNumber-1,letters.index(coorLetter)+k])
            elif orientation == 'down':
                for k in range (4):
                    OptionalShips[j].append([coorNumber-1+k,letters.index(coorLetter)])

        if shipType == 'P':
            if orientation == 'right':
                for k in range (2):
                    OptionalShips[j].append([coorNumber-1,letters.index(coorLetter)+k])
            elif orientation == 'down':
                for k in range (2):
                    OptionalShips[j].append([coorNumber-1+k,letters.index(coorLetter)])

        j += 1
```

There is two battleship and four patroal boat so we create lists for them after that we collect them all in a list to make it easier to reach.

First we create list for letters to convert them numbers to get indexes.

It looks like that: 

```python
letters = ['A','B','C','D','E','F','G','H','I','J']
```

(We may use dictionary which we will use later, but i could not think that.)

shipType = B,P

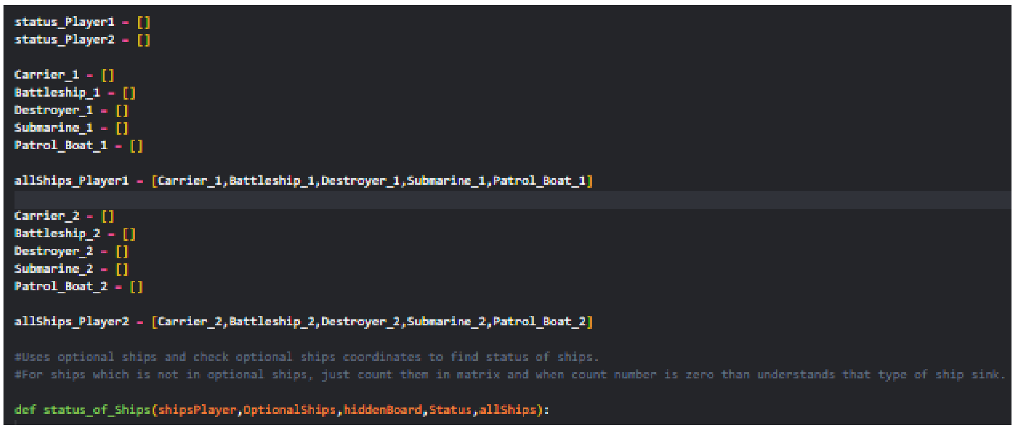
orientation = right,down

coorLetter = Horizontal vector areas letters (A,B,C,D,E,F,G,H,I,J)

coorNumber = Vertical vector areas numbers (1,2,3,4,5,6,7,8,9,10)

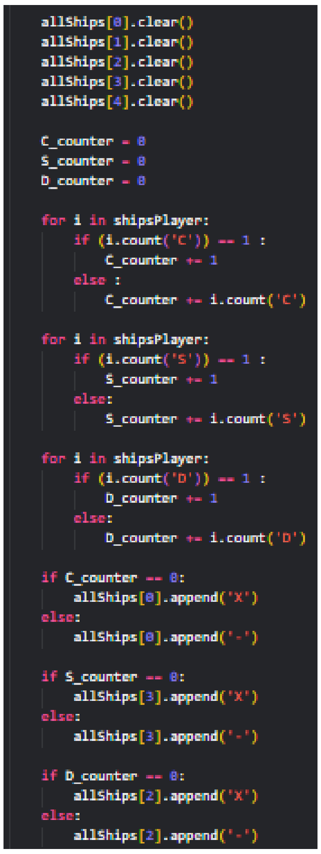
This function use the shipType to find shipType, use orientation to find direction and find ship coordinates by doing start from (coorLetter,coorNumber) coordinate and goes orientation directon for shipType's length. (Battlesip length is 4 area and Patroal Boat length is 2 area.)

Finding Status of ships (If they still float or sinked)



```python
status_Player1 = []
status_Player2 = []

Carrier_1 = []
Battleship_1 = []
Destroyer_1 = []
Submarine_1 = []
Patrol_Boat_1 = []

allShips_Player1 = [Carrier_1,Battleship_1,Destroyer_1,Submarine_1,Patrol_Boat_1]

Carrier_2 = []
Battleship_2 = []
Destroyer_2 = []
Submarine_2 = []
Patrol_Boat_2 = []

allShips_Player2 = [Carrier_2,Battleship_2,Destroyer_2,Submarine_2,Patrol_Boat_2]

#Uses optional ships and check optional ships coordinates to find status of ships.
#For ships which is not in optional ships, just count them in matrix and when count number is zero than understands that type of ship sink.

def status_of_Ships(shipsPlayer,OptionalShips,hiddenBoard,Status,allShips):
```

First we create lists for every ship type for each player and bring them together in sllShips_Player list to reach them easily. At the end of the function we will append ship status to status_Player lists.



```
allShips[0].clear()
allShips[1].clear()
allShips[2].clear()
allShips[3].clear()
allShips[4].clear()

C_counter = 0
S_counter = 0
D_counter = 0

for i in shipsPlayer:
    if (i.count('C')) == 1 :
        C_counter += 1
    else :
        C_counter += i.count('C')

for i in shipsPlayer:
    if (i.count('S')) == 1 :
        S_counter += 1
    else:
        S_counter += i.count('S')

for i in shipsPlayer:
    if (i.count('D')) == 1 :
        D_counter += 1
    else:
        D_counter += i.count('D')

if C_counter == 0:
    allShips[0].append('X')
else:
    allShips[0].append('-')

if S_counter == 0:
    allShips[3].append('X')
else:
    allShips[3].append('-')

if D_counter == 0:
    allShips[2].append('X')
else:
    allShips[2].append('-')
```

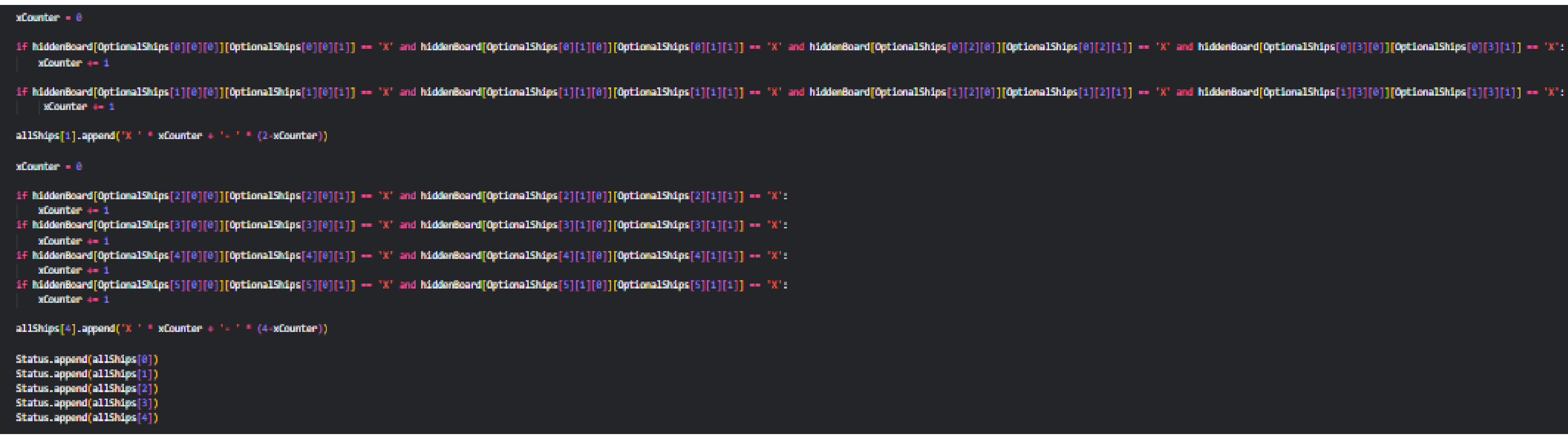
In this function we first clear all lists in allShip list becasue we will use this function recursively in game() function so every time this function runs it clears allShip lists to clearly check if any ship sinked.

This function has 2 part.

In first part function checks for C, D and S ships. Because they have one ship on board so we just count areas filled with C, D or S. For each C,D and S function checks horizontaly and verticaly then when there is no area filled with C,D or S symbols anymore. It means theese areas shooted and ship symbols replaced with 'X' with hit sign. Let say all 'D' ships hitted and no 'S hitted then function will append 'X' to Destroyer_playernumber list and '-' to Submarine_playernumber list. So we will use this X symbols and – symbols to print all ship status.

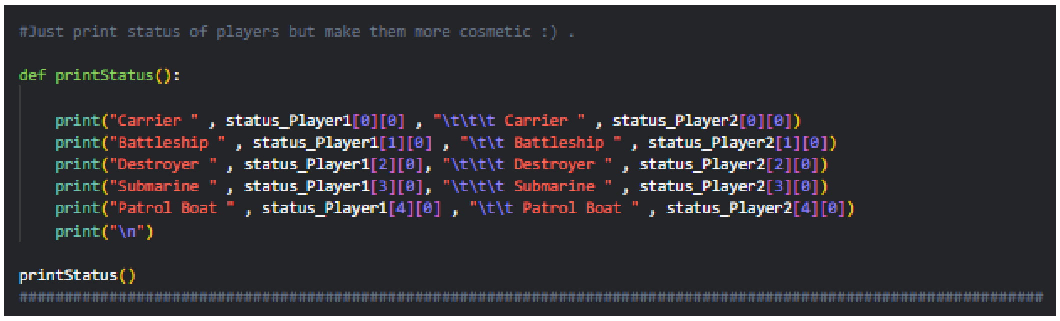
In second part function checks coordinates we get from optionalplayer files. Looks for how many Battleship and Patroal boat hitted then append them Battleship_playernumber or Patroal_Boat_playernumber list.

After check every ship type, append their status to status_Player list.



```
xCounter = 0

if hiddenBoard[OptionalShips[0][0][0]][OptionalShips[0][0][1]] == 'X' and hiddenBoard[OptionalShips[0][1][0]][OptionalShips[0][1][1]] == 'X' and hiddenBoard[OptionalShips[0][2][0]][OptionalShips[0][2][1]] == 'X' and hiddenBoard[OptionalShips[0][3][0]][OptionalShips[0][3][1]] == 'X':
    xCounter += 1

if hiddenBoard[OptionalShips[1][0][0]][OptionalShips[1][0][1]] == 'X' and hiddenBoard[OptionalShips[1][1][0]][OptionalShips[1][1][1]] == 'X' and hiddenBoard[OptionalShips[1][2][0]][OptionalShips[1][2][1]] == 'X' and hiddenBoard[OptionalShips[1][3][0]][OptionalShips[1][3][1]] == 'X':
     xCounter += 1

allShips[1].append('X ' * xCounter + '- ' * (2-xCounter))

xCounter = 0

if hiddenBoard[OptionalShips[2][0][0]][OptionalShips[2][0][1]] == 'X' and hiddenBoard[OptionalShips[2][1][0]][OptionalShips[2][1][1]] == 'X':
    xCounter += 1
if hiddenBoard[OptionalShips[3][0][0]][OptionalShips[3][0][1]] == 'X' and hiddenBoard[OptionalShips[3][1][0]][OptionalShips[3][1][1]] == 'X':
    xCounter += 1
if hiddenBoard[OptionalShips[4][0][0]][OptionalShips[4][0][1]] == 'X' and hiddenBoard[OptionalShips[4][1][0]][OptionalShips[4][1][1]] == 'X':
    xCounter += 1
if hiddenBoard[OptionalShips[5][0][0]][OptionalShips[5][0][1]] == 'X' and hiddenBoard[OptionalShips[5][1][0]][OptionalShips[5][1][1]] == 'X':
    xCounter += 1

allShips[4].append('X ' * xCounter + '- ' * (4-xCounter))

Status.append(allShips[0])
Status.append(allShips[1])
Status.append(allShips[2])
Status.append(allShips[3])
Status.append(allShips[4])
```

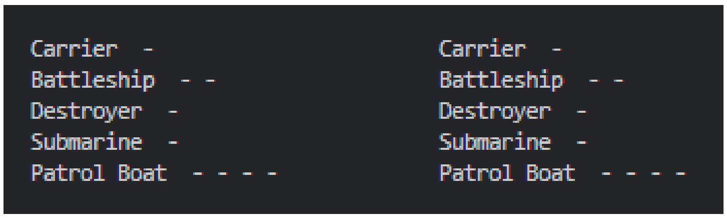
Printing Status of ships



```python
#Just print status of players but make them more cosmetic :) .

def printStatus():

    print("Carrier " , status_Player1[0][0] , "\t\t\t Carrier " , status_Player2[0][0])
    print("Battleship " , status_Player1[1][0] , "\t\t Battleship " , status_Player2[1][0])
    print("Destroyer " , status_Player1[2][0], "\t\t\t Destroyer " , status_Player2[2][0])
    print("Submarine " , status_Player1[3][0], "\t\t\t Submarine " , status_Player2[3][0])
    print("Patrol Boat " , status_Player1[4][0] , "\t\t Patrol Boat " , status_Player2[4][0])
    print("\n")

printStatus()
########################################################################################################################
```

This part just print ship name and ship’s status for each player.

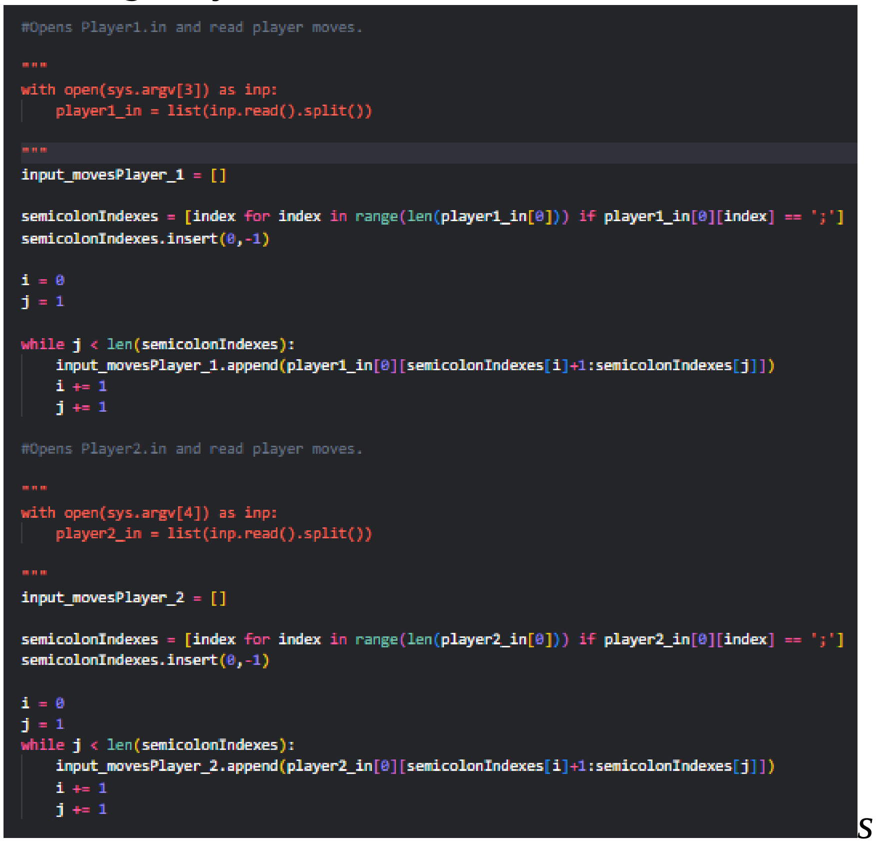
Output will be like that:



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

Second Part - Reading Player Moves and Organize Them

Reading Player Move



```
#Opens Player1.in and read player moves.

"""
with open(sys.argv[3]) as inp:
    player1_in = list(inp.read().split())

"""
input_movesPlayer_1 = []

semicolonIndexes = [index for index in range(len(player1_in[0])) if player1_in[0][index] == ';']
semicolonIndexes.insert(0,-1)

i = 0
j = 1

while j < len(semicolonIndexes):
    input_movesPlayer_1.append(player1_in[0][semicolonIndexes[i]+1:semicolonIndexes[j]])
    i += 1
    j += 1

#Opens Player2.in and read player moves.

"""
with open(sys.argv[4]) as inp:
    player2_in = list(inp.read().split())

"""
input_movesPlayer_2 = []

semicolonIndexes = [index for index in range(len(player2_in[0])) if player2_in[0][index] == ';']
semicolonIndexes.insert(0,-1)

i = 0
j = 1
while j < len(semicolonIndexes):
    input_movesPlayer_2.append(player2_in[0][semicolonIndexes[i]+1:semicolonIndexes[j]])
    i += 1
    j += 1
```

s

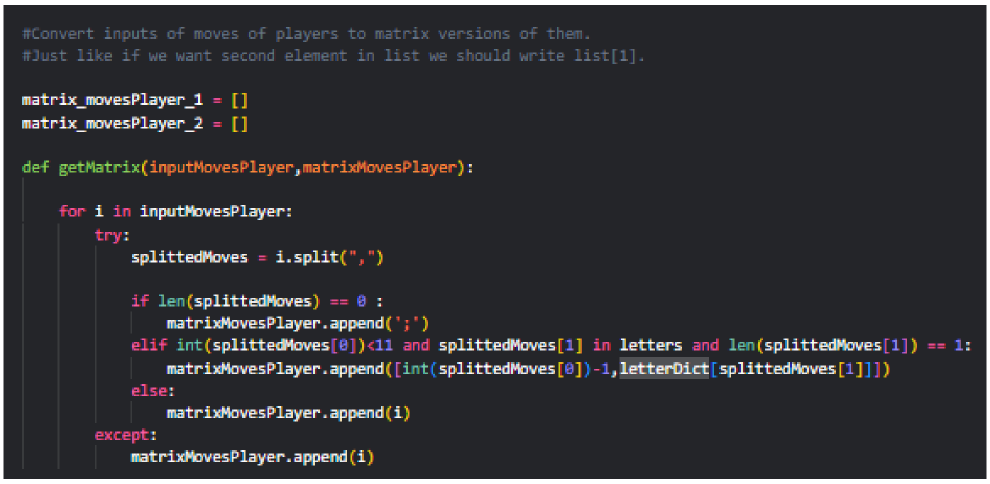
Reads player.in files and separate them when there is a semicolon.

For example:

5,E;10,G;8,I;4,C;8,F;4,F   ,   Player.in file example

(5,E), (10,G), (8,I), (4,C), (8,F), (4,F)   ,   input_movesPlayer list example

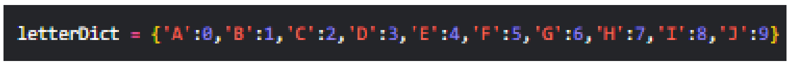
Convert Player Inputs to Matrix Version of Them



```python
#Convert inputs of moves of players to matrix versions of them.
#Just like if we want second element in list we should write list[1].

matrix_movesPlayer_1 = []
matrix_movesPlayer_2 = []

def getMatrix(inputMovesPlayer,matrixMovesPlayer):

    for i in inputMovesPlayer:
        try:
            splittedMoves = i.split(",")

            if len(splittedMoves) == 0 :
                matrixMovesPlayer.append(';')
            elif int(splittedMoves[0])<11 and splittedMoves[1] in letters and len(splittedMoves[1]) == 1:
                matrixMovesPlayer.append([int(splittedMoves[0])-1,letterDict[splittedMoves[1]]])
            else:
                matrixMovesPlayer.append(i)
        except:
            matrixMovesPlayer.append(i)
```

This function take input_movesPlayer lists and convert them to matrix versions by decrease one to verticals element and using Dictionary for horizontaly Letter elements.

Dictionary:



```python
letterDict = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7,'I':8,'J':9}
```

We do that because list[0] actually first element of list. S oto get matrix version of move, we just decrease on efor (x,y) so it becomes (x-1,y-1).

If input is wrong (like 11,A;55,M;A,A; .. etc) function append that input as string so this input:

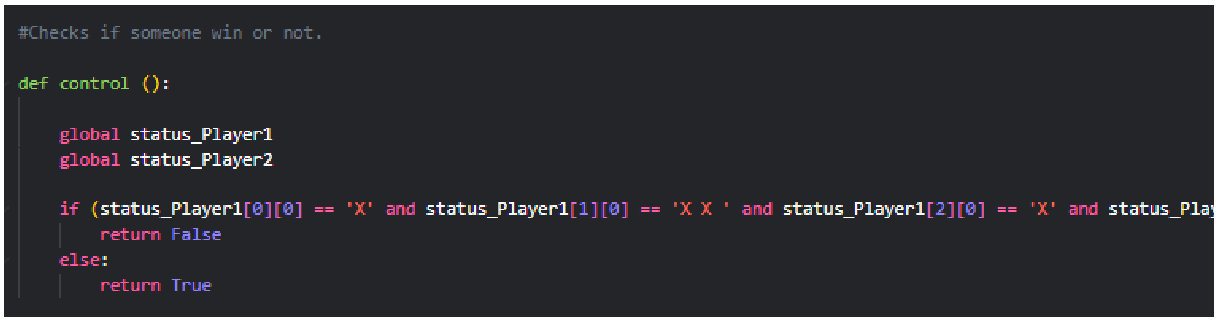
5,A;FF,FF;11,Z;7,J;

Will be look like that in matrixMovesPlayer list:

(5,A),('FF,FF'),('11,Z'),(7,J)

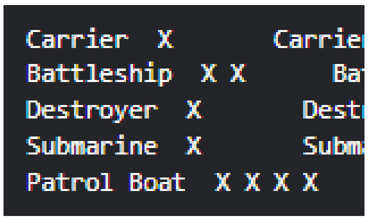
Final Part – Game() function and Control() function

Control() Function



```
#Checks if someone win or not.

def control ():

    global status_Player1
    global status_Player2

    if (status_Player1[0][0] == 'X' and status_Player1[1][0] == 'X X ' and status_Player1[2][0] == 'X' and status_Play
        return False
    else:
        return True
```

This function basically just checks if printstatus() functions output look like that:



```
Carrier  X       Carrie
Battleship  X X      Ba
Destroyer  X       Dest
Submarine  X       Subm
Patrol Boat  X X X X
```

Control function checks every ship to understand if that ship sinked or not. If all ships sinked then function return False, otherwise function return true.

In game function game() runs while control() returning True.

Game() Function

```
print("Battle of Ships Game\n\n")
```

```
roundCounter = 1
moveCounter_Player1 = 0
moveCounter_Player2 = 0
```

```
#Game

def game():
    try:
        shipNames = ['B','C','D','S','P']

        global roundCounter
        global moveCounter_Player1
        global moveCounter_Player2

        while control() and ((len(input_movesPlayer_1) >= moveCounter_Player1) and (len(input_movesPlayer_2) >= moveCounter_Player2)):
            if (len(input_movesPlayer_1) == moveCounter_Player1) and (len(input_movesPlayer_2) == moveCounter_Player2):
                raise IndexError
            if roundCounter % 2 == 0:
                player = "Player2"

            else:
                player = "Player1"

            print("\n")
            print(player + "'s Move\n\nRound : ",   roundCounter//2,"\t\t\t\t Grid Size: " + str(len(letters)) + "x" + str(len(letters)) + "\n")

            print_hiddenBoard()

            status_of_Ships(shipsPlayer_1,OptionalShips_Player1,hiddenBoard_Player1,status_Player1,allShips_Player1)
            status_of_Ships(shipsPlayer_2,OptionalShips_Player2,hiddenBoard_Player2,status_Player2,allShips_Player2)

            printStatus()
```

We print “Battle of Ships Game” outside of function because otherwise it will be printed in every round, and we do not want this.

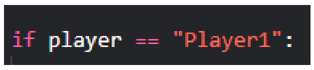
roundCounter, counts rounds.Player1 starts game. Funticon decide whis player moves by rounCounter if roundCounter is odd number that means Player2 is playing, else Player1 is moving.

MoveCounterPlayers just count what was last moves index of player. Because if player wrote wrong move in input file (like already hitted coordinates,or like 11,A;45,BJ; … etc.) we just print error mesage and take next input which comes after last wrong move input of same player. That’s why we count players move counters seperately.

After that we run this function while control() is true and two players still have moves. If one of players move finish. Code raise IndexError and in IndexError we print “Player left, kaBOOM: run for your life!”

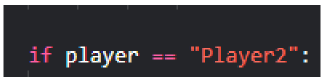
```
print("\n")
print(player,'left.\n\n')
print("kaBOOM: run for your life!")
```

Then code checks which player is playing



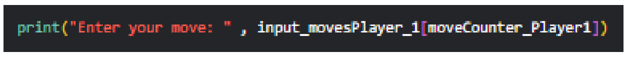
```
if player == "Player1":
```

.

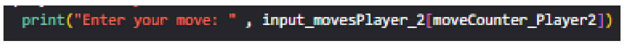


```
if player == "Player2":
```

After understands which player is moving gets input of player.

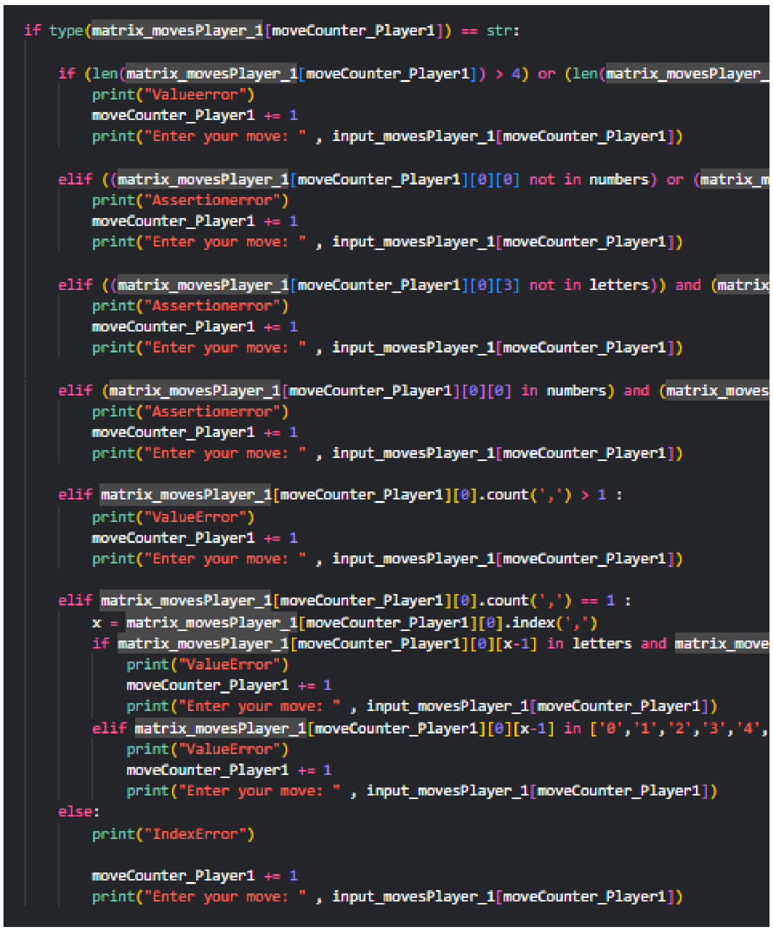


```
print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])
```



```
print("Enter your move: " , input_movesPlayer_2[moveCounter_Player2])
```

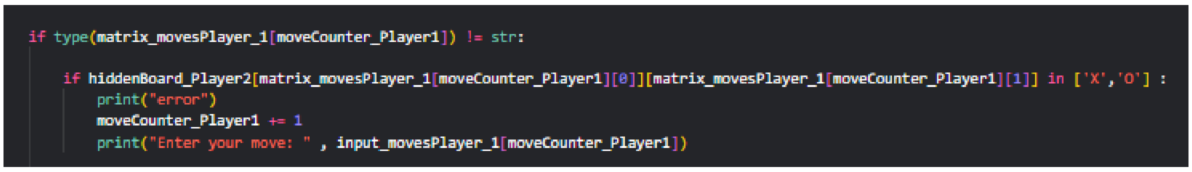
If input is string in matrixMovesPlayer list function decides which type of error is this and print it, after that next move of same player.



```
if type(matrix_movesPlayer_1[moveCounter_Player1]) == str:

    if (len(matrix_movesPlayer_1[moveCounter_Player1]) > 4) or (len(matrix_movesPlayer_
        print("Valueerror")
        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])

    elif ((matrix_movesPlayer_1[moveCounter_Player1][0][0] not in numbers) or (matrix_m
        print("Assertionerror")
        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])

    elif ((matrix_movesPlayer_1[moveCounter_Player1][0][3] not in letters)) and (matrix
        print("Assertionerror")
        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])

    elif (matrix_movesPlayer_1[moveCounter_Player1][0][0] in numbers) and (matrix_moves
        print("Assertionerror")
        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])

    elif matrix_movesPlayer_1[moveCounter_Player1][0].count(',') > 1 :
        print("ValueError")
        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])

    elif matrix_movesPlayer_1[moveCounter_Player1][0].count(',') == 1 :
        x = matrix_movesPlayer_1[moveCounter_Player1][0].index(',')
        if matrix_movesPlayer_1[moveCounter_Player1][0][x-1] in letters and matrix_move
            print("ValueError")
            moveCounter_Player1 += 1
            print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])
        elif matrix_movesPlayer_1[moveCounter_Player1][0][x-1] in ['0','1','2','3','4',
            print("ValueError")
            moveCounter_Player1 += 1
            print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])
    else:
        print("IndexError")

        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])
```

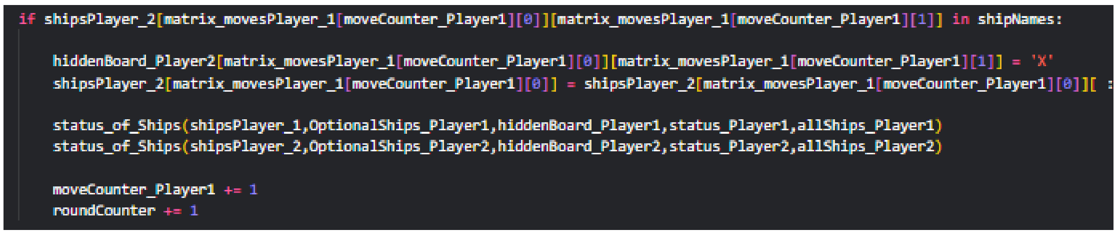
(It is for player 1 but for player 2 is same.)

If move is not string in matrixMovesPlayer list then program checks if moves matrix (x,y) is shooted already. If that area is hitted then print error and take next move.



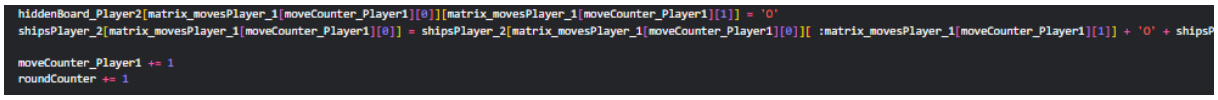
```
if type(matrix_movesPlayer_1[moveCounter_Player1]) != str:

    if hiddenBoard_Player2[matrix_movesPlayer_1[moveCounter_Player1][0]][matrix_movesPlayer_1[moveCounter_Player1][1]] in ['X','O'] :
        print("error")
        moveCounter_Player1 += 1
        print("Enter your move: " , input_movesPlayer_1[moveCounter_Player1])
```

If move is not string and move's coordinates is not shooted before then program checks if there is a ship or not wtih using shipNames list. If there is a ship, then index of it will be changed as 'X' in hiddenBoard and shipPlayer lists.



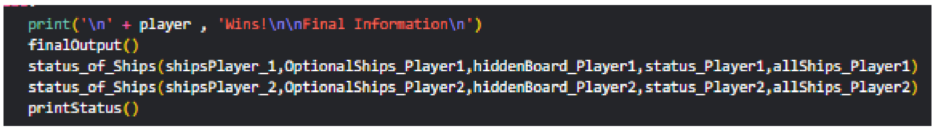
```
if shipsPlayer_2[matrix_movesPlayer_1[moveCounter_Player1][0]][matrix_movesPlayer_1[moveCounter_Player1][1]] in shipNames:

    hiddenBoard_Player2[matrix_movesPlayer_1[moveCounter_Player1][0]][matrix_movesPlayer_1[moveCounter_Player1][1]] = 'X'
    shipsPlayer_2[matrix_movesPlayer_1[moveCounter_Player1][0]] = shipsPlayer_2[matrix_movesPlayer_1[moveCounter_Player1][0]][ :

    status_of_Ships(shipsPlayer_1,OptionalShips_Player1,hiddenBoard_Player1,status_Player1,allShips_Player1)
    status_of_Ships(shipsPlayer_2,OptionalShips_Player2,hiddenBoard_Player2,status_Player2,allShips_Player2)

    moveCounter_Player1 += 1
    roundCounter += 1
```

If there is no ship, then index of it will be changed as 'O' in hiddenBoard and shipPlayer lists.



```
hiddenBoard_Player2[matrix_movesPlayer_1[moveCounter_Player1][0]][matrix_movesPlayer_1[moveCounter_Player1][1]] = 'O'
shipsPlayer_2[matrix_movesPlayer_1[moveCounter_Player1][0]] = shipsPlayer_2[matrix_movesPlayer_1[moveCounter_Player1][0]][ :matrix_movesPlayer_1[moveCounter_Player1][1]] + 'O' + shipsP

moveCounter_Player1 += 1
roundCounter += 1
```

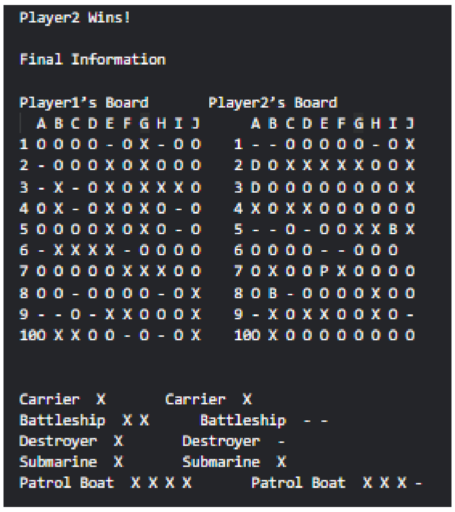
After playing all rounds if one player wins then program will print:



```
print('\n' + player , 'Wins!\n\nFinal Information\n')
finalOutput()
status_of_Ships(shipsPlayer_1,OptionalShips_Player1,hiddenBoard_Player1,status_Player1,allShips_Player1)
status_of_Ships(shipsPlayer_2,OptionalShips_Player2,hiddenBoard_Player2,status_Player2,allShips_Player2)
printStatus()
```

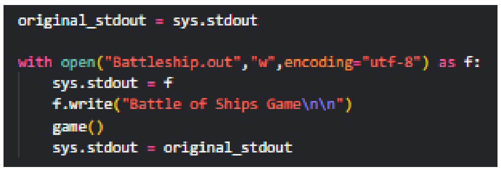
Before winning situation we alway printed hidden board because we did not want to show all ships on board but end of the game (if someone won) we call finalOutput() function to show players (who won) tos how ships that looser one could not shoot.

Output will be like that:



```
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 -
100 X X O O - O - O X    100 X O 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 -
```

And finally after program finished we print outputs and write them in Battleship.out file.



```
original_stdout = sys.stdout

with open("Battleship.out","w",encoding="utf-8") as f:
    sys.stdout = f
    f.write("Battle of Ships Game\n\n")
    game()
    sys.stdout = original_stdout
```

Analysis

The code written in 4 days so it has problems. Like there is n need letters list usage in pull_OptionalShips() function, Optional ship file in B1,B2,P1,P2,P3,P4 order otherwise (like if you wrtie first Patroal Boats.) It will not work right. In game() function detect errors part for moves, i could not test it well so probably there is some mistakes especially about Index error (maybe there is no mistake but like i said i could not test it well).