BBM103 ASSIGNMENT 4 BATTLE OF SHIPS

ŞÜKRİYE ÖZTÜRK
HACETTEPE UNIVERSITY 2210356110

01.01.2023

CONTENTS

Analysis…………………………………………………………………….………….…2

Design……………………………………………………. …………………………..3

Pseudocode……………………………………………………………………11

Programmer's Catalogue……………………………………………………………. 12

User Catalogue……………………...…………………………...…………………………..13

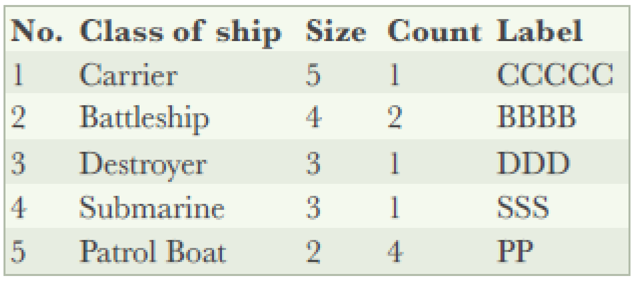
Tutorial (User's Manual) ……….……………………………………………..………. 13

Restrictions…………………………………………………………..….…15

Analysis:

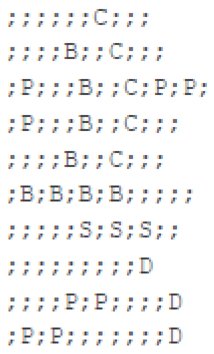
Battleship is a game which based on sinking the other player's ships. This game is played with 2 players. At first, each player places their ships to a hidden board which other player cannot see. Then, each player tries to guess the coordinates of opponent's ships. Both players give a coordinate every turn. The first player who finds coordinates of all ships of other player and shots them wins the game. In other case, game finishes draw.

Players place their ships with writing the coordinates of ships to a text file. The boards are 10x10 squares and every square can take maximum one part of a ship. The categories of ships are given below.



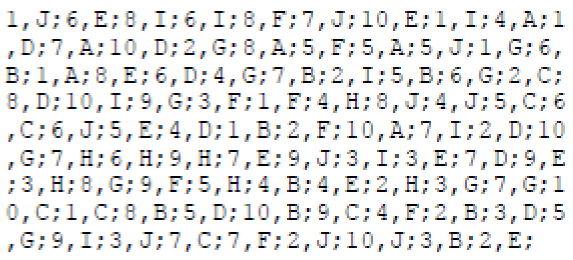
No.	Class of ship	Size	Count	Label
1	Carrier	5	1	CCCCC
2	Battleship	4	2	BBBB
3	Destroyer	3	1	DDD
4	Submarine	3	1	SSS
5	Patrol Boat	2	4	PP

For example, *"carrier"* class ship consists of 5 parts. Each player should place 1 carrier ship to their board. Player should place their ships to text file with *";, C, B, D, S, P "* characters. Each *";"* character should represent a grid of board. There is an example of an input file 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
```

Also, each player gives the positions which they want to shot with an input file. Every position should given as *"row,column"*. Rows should be represented with numbers and columns should be represented with letters. So a coordinate should be written as, *"number,letter"*. Each position should be separated with *";"*. There is an example of input file below.

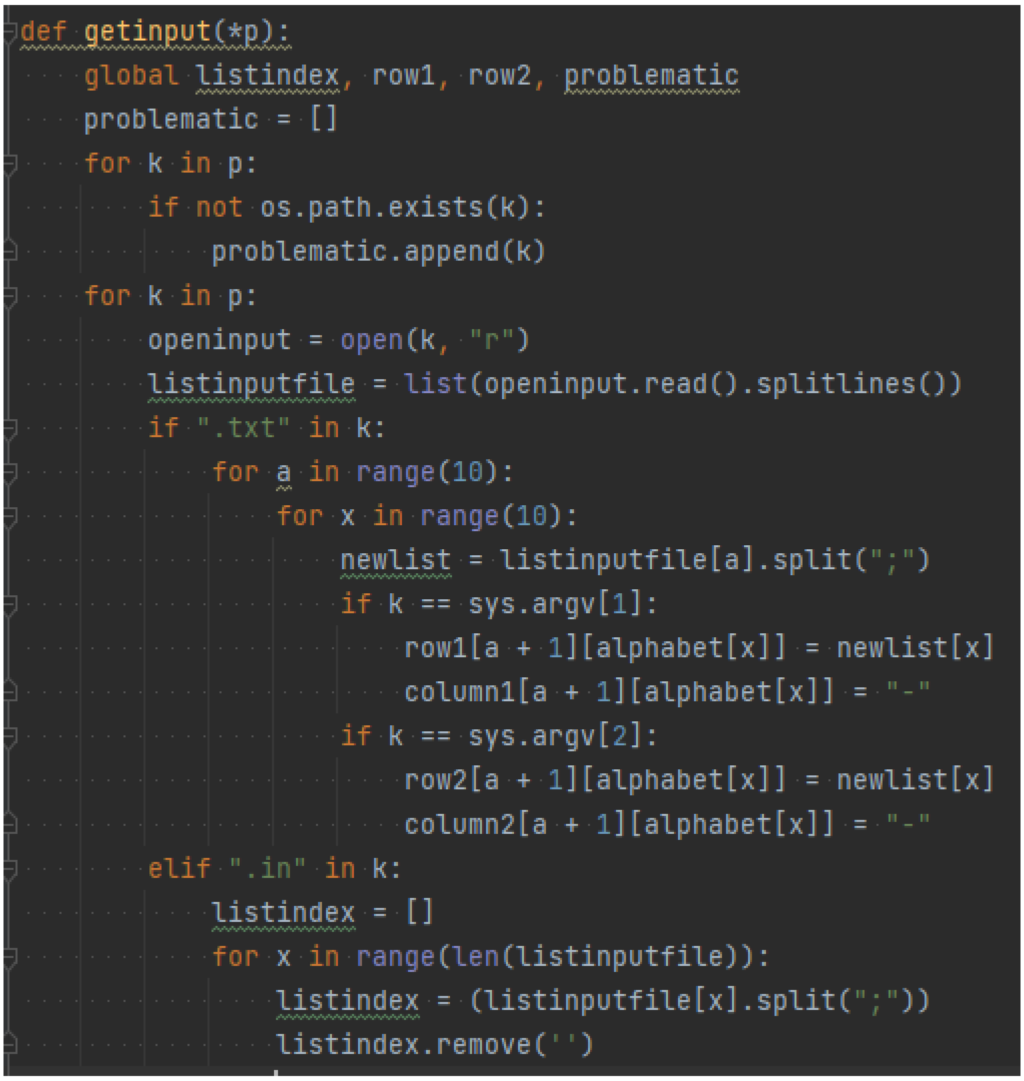


```
1,J;6,E;8,I;6,I;8,F;7,J;10,E;1,I;4,A;1
,D;7,A;10,D;2,G;8,A;5,F;5,A;5,J;1,G;6,
B;1,A;8,E;6,D;4,G;7,B;2,I;5,B;6,G;2,C;
8,D;10,I;9,G;3,F;1,F;4,H;8,J;4,J;5,C;6
,C;6,J;5,E;4,D;1,B;2,F;10,A;7,I;2,D;10
,G;7,H;6,H;9,H;7,E;9,J;3,I;3,E;7,D;9,E
;3,H;8,G;9,F;5,H;4,B;4,E;2,H;3,G;7,G;1
0,C;1,C;8,B;5,D;10,B;9,C;4,F;2,B;3,D;5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

Design:

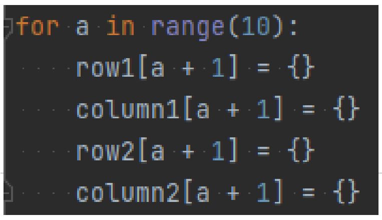
I started to code with creating functions. At first, I just created *"play1, play2, getinput, getoutput"* functions. As code developed, I've needed new functions as *"ships, board"*.

Function *getinput(*p)*:



```
def getinput(*p):
    global listindex, row1, row2, problematic
    problematic = []
    for k in p:
        if not os.path.exists(k):
            problematic.append(k)
    for k in p:
        openinput = open(k, "r")
        listinputfile = list(openinput.read().splitlines())
        if ".txt" in k:
            for a in range(10):
                for x in range(10):
                    newlist = listinputfile[a].split(";")
                    if k == sys.argv[1]:
                        row1[a + 1][alphabet[x]] = newlist[x]
                        column1[a + 1][alphabet[x]] = "-"
                    if k == sys.argv[2]:
                        row2[a + 1][alphabet[x]] = newlist[x]
                        column2[a + 1][alphabet[x]] = "-"
        elif ".in" in k:
            listindex = []
            for x in range(len(listinputfile)):
                listindex = (listinputfile[x].split(";"))
                listindex.remove('')
```

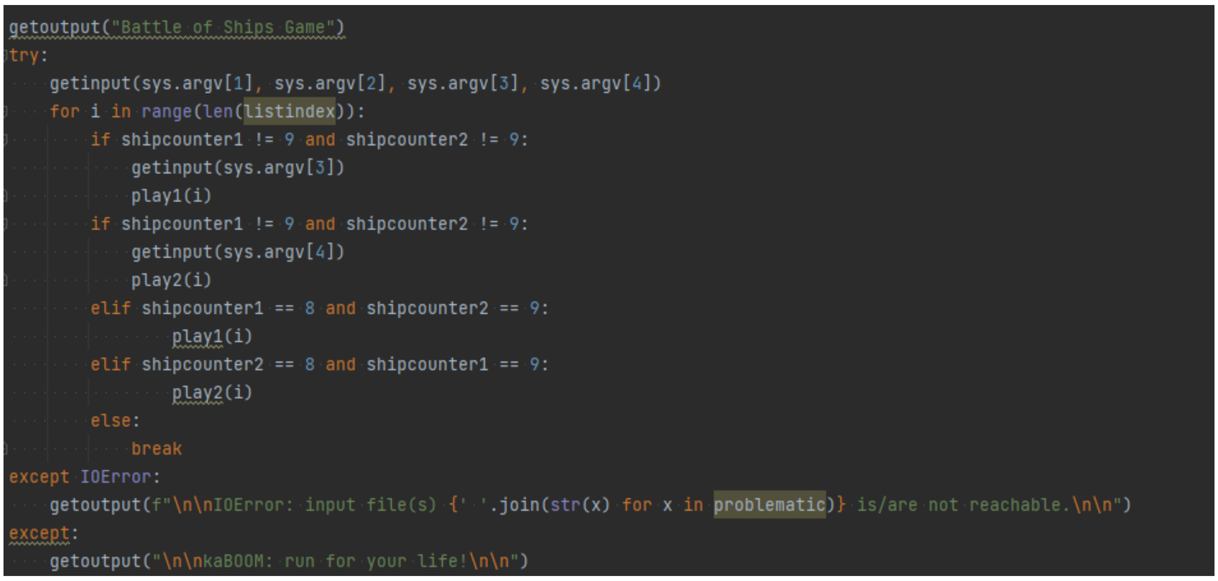
This function provides the information inside the file which is given. If the argument is shipboard file, function splits elements from *";"* and takes the letters between them. At the end, the ships become the values of *"row*" dictionaries.



```
for a in range(10):
    row1[a + 1] = {}
    column1[a + 1] = {}
    row2[a + 1] = {}
    column2[a + 1] = {}
```

A board consists of 10 rows so, I created 10 different dictionaries in dictionaries for every row of a player. Every part of a ship is a value for *row[a+1]*. While placing the ships, I also created *"column"* dictionaries for player's hidden boards. For this command, column dictionaries just have *"-"* as values.

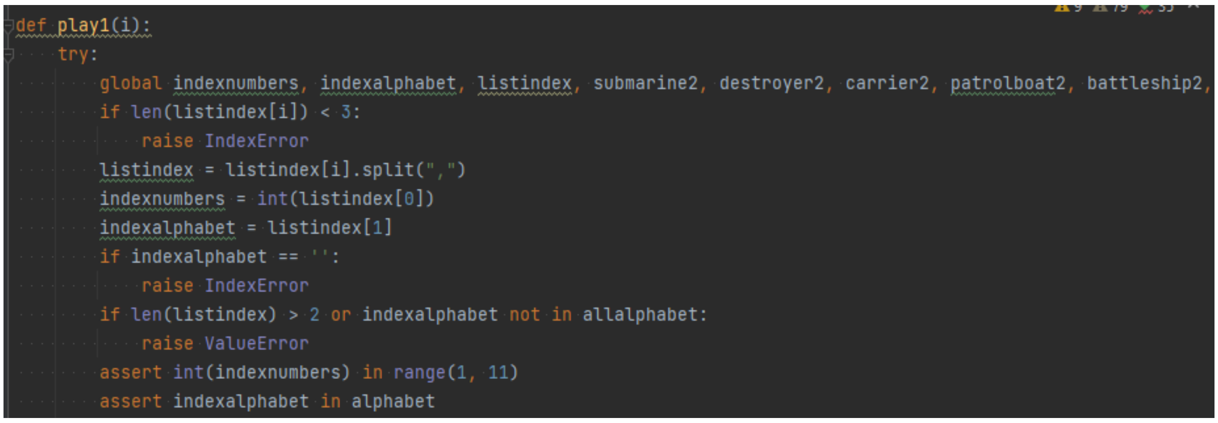
If the given argument belongs to player1, values placing to *column1* and *row1*. Otherwise, values placing to *row2* and *column2*. Also I created a *problematic* list for the arguments which is going to give *IOError*. If the file doesn't exist at path, *sys.argv* is being added to *problematic* list.



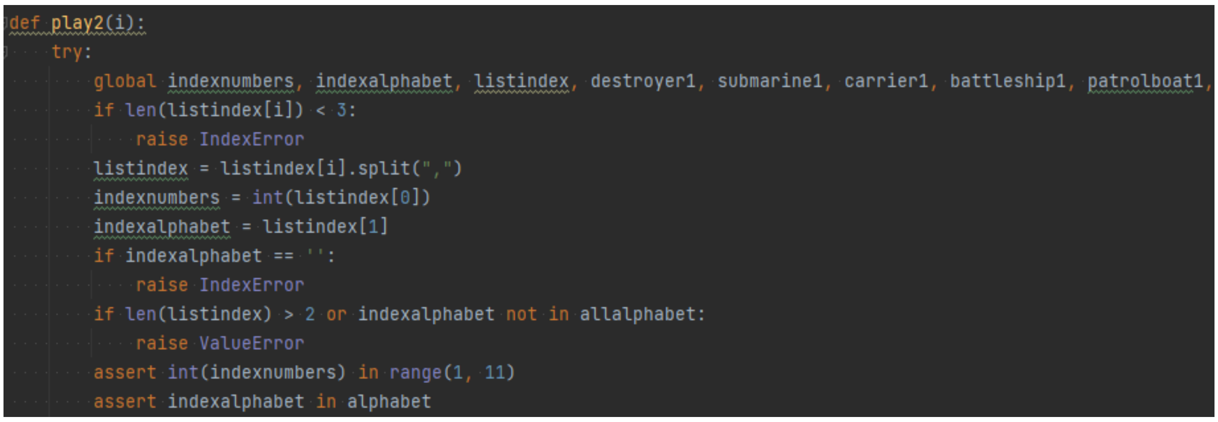
```python
getoutput("Battle of Ships Game")
try:
    getinput(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
    for i in range(len(listindex)):
        if shipcounter1 != 9 and shipcounter2 != 9:
            getinput(sys.argv[3])
            play1(i)
        if shipcounter1 != 9 and shipcounter2 != 9:
            getinput(sys.argv[4])
            play2(i)
        elif shipcounter1 == 8 and shipcounter2 == 9:
                play1(i)
        elif shipcounter2 == 8 and shipcounter1 == 9:
                play2(i)
        else:
            break
except IOError:
    getoutput(f"\n\nIOError: input file(s) {' '.join(str(x) for x in problematic)} is/are not reachable.\n\n")
except:
    getoutput("\n\nkaBOOM: run for your life!\n\n")
```

As can be seen, *problematic* list created in case *IOError* occurs.

Function *play1(i)* and *play2(i)*:



```python
def play1(i):
    try:
        global indexnumbers, indexalphabet, listindex, submarine2, destroyer2, carrier2, patrolboat2, battleship2,
        if len(listindex[i]) < 3:
            raise IndexError
        listindex = listindex[i].split(",")
        indexnumbers = int(listindex[0])
        indexalphabet = listindex[1]
        if indexalphabet == '':
            raise IndexError
        if len(listindex) > 2 or indexalphabet not in allalphabet:
            raise ValueError
        assert int(indexnumbers) in range(1, 11)
        assert indexalphabet in alphabet
```

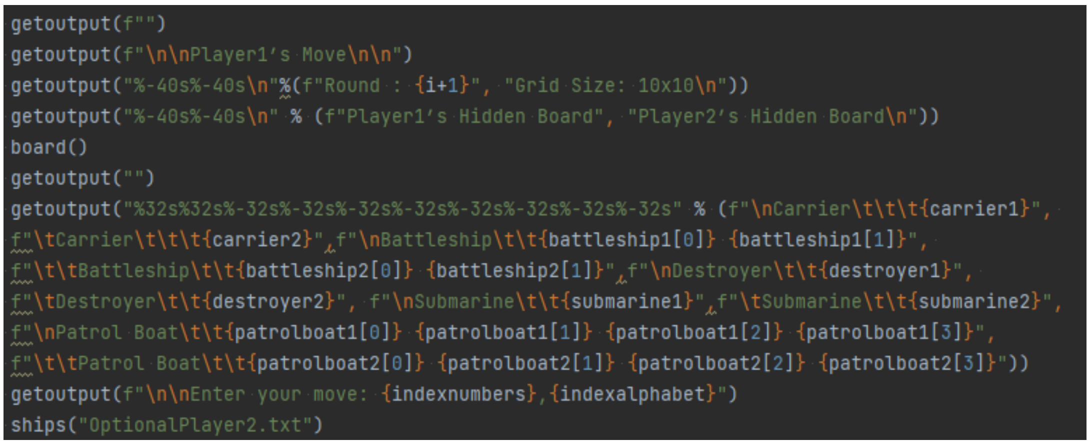


```python
def play2(i):
    try:
        global indexnumbers, indexalphabet, listindex, destroyer1, submarine1, carrier1, battleship1, patrolboat1,
        if len(listindex[i]) < 3:
            raise IndexError
        listindex = listindex[i].split(",")
        indexnumbers = int(listindex[0])
        indexalphabet = listindex[1]
        if indexalphabet == '':
            raise IndexError
        if len(listindex) > 2 or indexalphabet not in allalphabet:
            raise ValueError
        assert int(indexnumbers) in range(1, 11)
        assert indexalphabet in alphabet
```

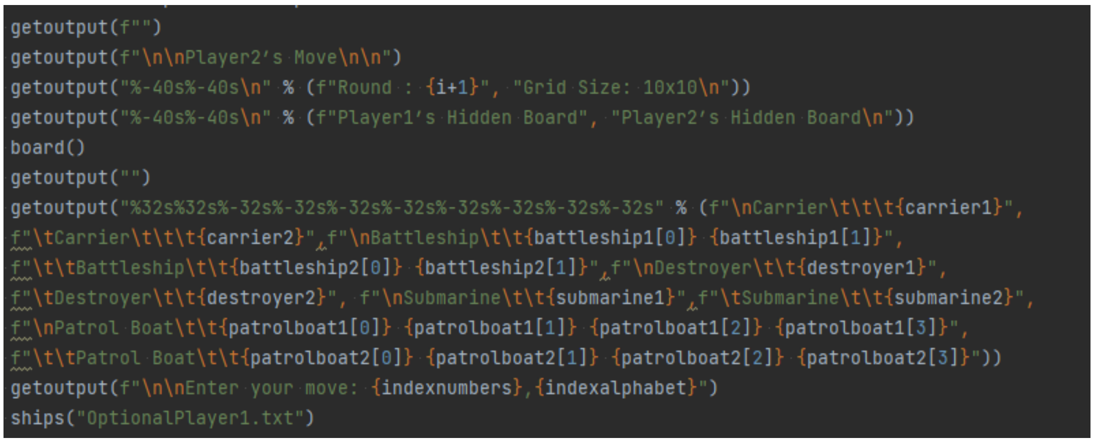
I wanted to analyse them at the same time because both functions are created at the same base.

At the beginning of the function, I’m checking the Error occasions. *listindex* is the list of given input by each player. Iteration *i* loops along length of *listindex*. I raised *IndexError* if length of *listindex* less than 3 because that means one of the arguments is missing.

Then I split *listindex* to *“indexnumbers”* and *“indexalphabet”* to make it simpler. *indexnumbers* contains row numbers of coordinates and *indexalphabet* contains column letters of coordinates.

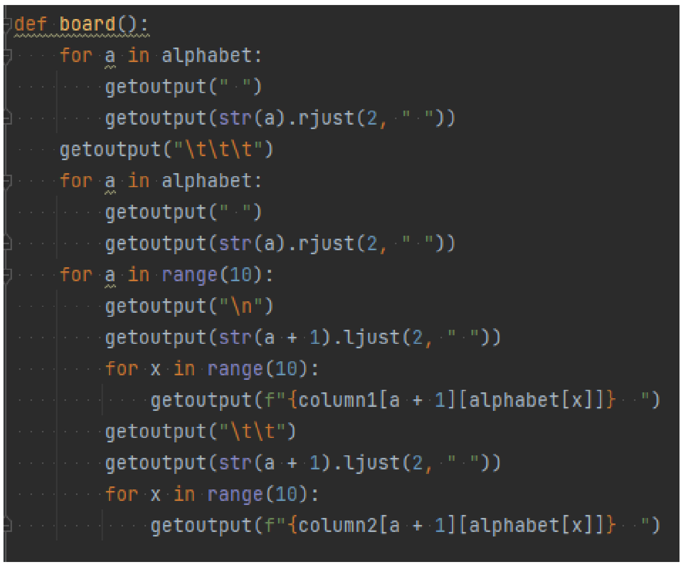


```python
getoutput(f"")
getoutput(f"\n\nPlayer1’s Move\n\n")
getoutput("%-40s%-40s\n"%(f"Round : {i+1}", "Grid Size: 10x10\n"))
getoutput("%-40s%-40s\n" % (f"Player1’s Hidden Board", "Player2’s Hidden Board\n"))
board()
getoutput("")
getoutput("%32s%32s%-32s%-32s%-32s%-32s%-32s%-32s%-32s%-32s" % (f"\nCarrier\t\t\t{carrier1}",
f"\tCarrier\t\t\t{carrier2}",f"\nBattleship\t\t{battleship1[0]} {battleship1[1]}",
f"\t\tBattleship\t\t{battleship2[0]} {battleship2[1]}",f"\nDestroyer\t\t{destroyer1}",
f"\tDestroyer\t\t{destroyer2}", f"\nSubmarine\t\t{submarine1}",f"\tSubmarine\t\t{submarine2}",
f"\nPatrol Boat\t\t{patrolboat1[0]} {patrolboat1[1]} {patrolboat1[2]} {patrolboat1[3]}",
f"\t\tPatrol Boat\t\t{patrolboat2[0]} {patrolboat2[1]} {patrolboat2[2]} {patrolboat2[3]}"))
getoutput(f"\n\nEnter your move: {indexnumbers},{indexalphabet}")
ships("OptionalPlayer2.txt")
```



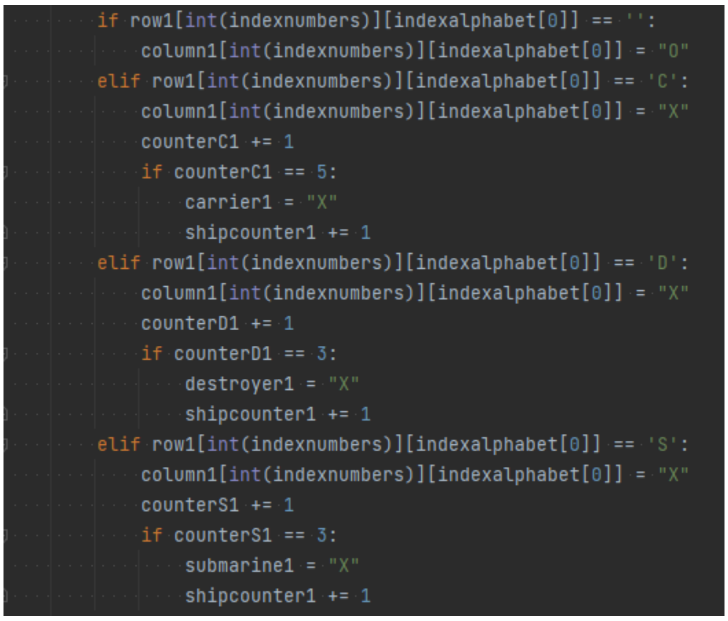
```python
getoutput(f"")
getoutput(f"\n\nPlayer2’s Move\n\n")
getoutput("%-40s%-40s\n" % (f"Round : {i+1}", "Grid Size: 10x10\n"))
getoutput("%-40s%-40s\n" % (f"Player1’s Hidden Board", "Player2’s Hidden Board\n"))
board()
getoutput("")
getoutput("%32s%32s%-32s%-32s%-32s%-32s%-32s%-32s%-32s%-32s" % (f"\nCarrier\t\t\t{carrier1}",
f"\tCarrier\t\t\t{carrier2}",f"\nBattleship\t\t{battleship1[0]} {battleship1[1]}",
f"\t\tBattleship\t\t{battleship2[0]} {battleship2[1]}",f"\nDestroyer\t\t{destroyer1}",
f"\tDestroyer\t\t{destroyer2}", f"\nSubmarine\t\t{submarine1}",f"\tSubmarine\t\t{submarine2}",
f"\nPatrol Boat\t\t{patrolboat1[0]} {patrolboat1[1]} {patrolboat1[2]} {patrolboat1[3]}",
f"\t\tPatrol Boat\t\t{patrolboat2[0]} {patrolboat2[1]} {patrolboat2[2]} {patrolboat2[3]}"))
getoutput(f"\n\nEnter your move: {indexnumbers},{indexalphabet}")
ships("OptionalPlayer1.txt")
```

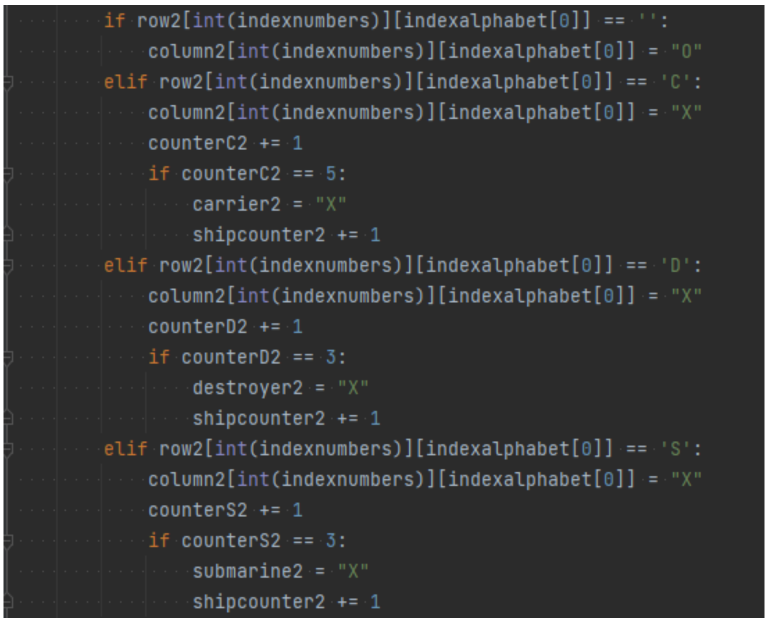
After Error check, I printed the hidden boards with such information.


```python
def board():
    for a in alphabet:
        getoutput(" ")
        getoutput(str(a).rjust(2, " "))
    getoutput("\t\t\t")
    for a in alphabet:
        getoutput(" ")
        getoutput(str(a).rjust(2, " "))
    for a in range(10):
        getoutput("\n")
        getoutput(str(a + 1).ljust(2, " "))
        for x in range(10):
            getoutput(f"{column1[a + 1][alphabet[x]]}  ")
        getoutput("\t\t")
        getoutput(str(a + 1).ljust(2, " "))
        for x in range(10):
            getoutput(f"{column2[a + 1][alphabet[x]]}  ")
```

Function *board():*

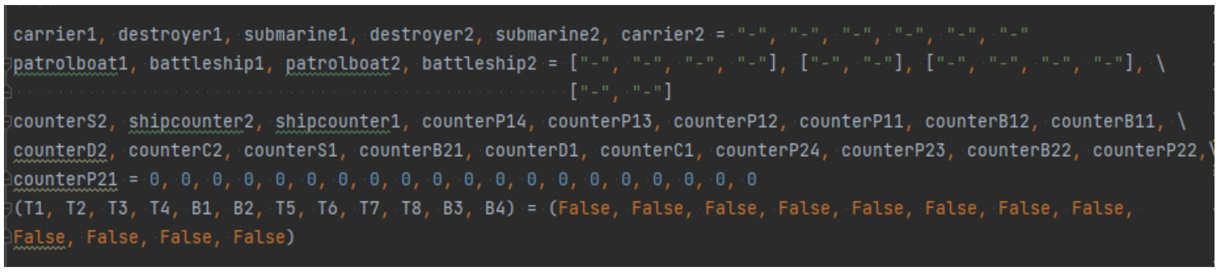
Prints the hidden boards for each of players. It prints column letters and then hidden board values of players for every row.


```python
        if row1[int(indexnumbers)][indexalphabet[0]] == '':
            column1[int(indexnumbers)][indexalphabet[0]] = "O"
        elif row1[int(indexnumbers)][indexalphabet[0]] == 'C':
            column1[int(indexnumbers)][indexalphabet[0]] = "X"
            counterC1 += 1
            if counterC1 == 5:
                carrier1 = "X"
                shipcounter1 += 1
        elif row1[int(indexnumbers)][indexalphabet[0]] == 'D':
            column1[int(indexnumbers)][indexalphabet[0]] = "X"
            counterD1 += 1
            if counterD1 == 3:
                destroyer1 = "X"
                shipcounter1 += 1
        elif row1[int(indexnumbers)][indexalphabet[0]] == 'S':
            column1[int(indexnumbers)][indexalphabet[0]] = "X"
            counterS1 += 1
            if counterS1 == 3:
                submarine1 = "X"
                shipcounter1 += 1
```


```python
        if row2[int(indexnumbers)][indexalphabet[0]] == '':
            column2[int(indexnumbers)][indexalphabet[0]] = "O"
        elif row2[int(indexnumbers)][indexalphabet[0]] == 'C':
            column2[int(indexnumbers)][indexalphabet[0]] = "X"
            counterC2 += 1
            if counterC2 == 5:
                carrier2 = "X"
                shipcounter2 += 1
        elif row2[int(indexnumbers)][indexalphabet[0]] == 'D':
            column2[int(indexnumbers)][indexalphabet[0]] = "X"
            counterD2 += 1
            if counterD2 == 3:
                destroyer2 = "X"
                shipcounter2 += 1
        elif row2[int(indexnumbers)][indexalphabet[0]] == 'S':
            column2[int(indexnumbers)][indexalphabet[0]] = "X"
            counterS2 += 1
            if counterS2 == 3:
                submarine2 = "X"
                shipcounter2 += 1
```

The next step is checking the singular ships. It checks the *row* dictionary if there is a ship, then changes the value depending on the value. The code gives value *"X"* to the column if there is a ship, otherwise gives the value *"O"*.

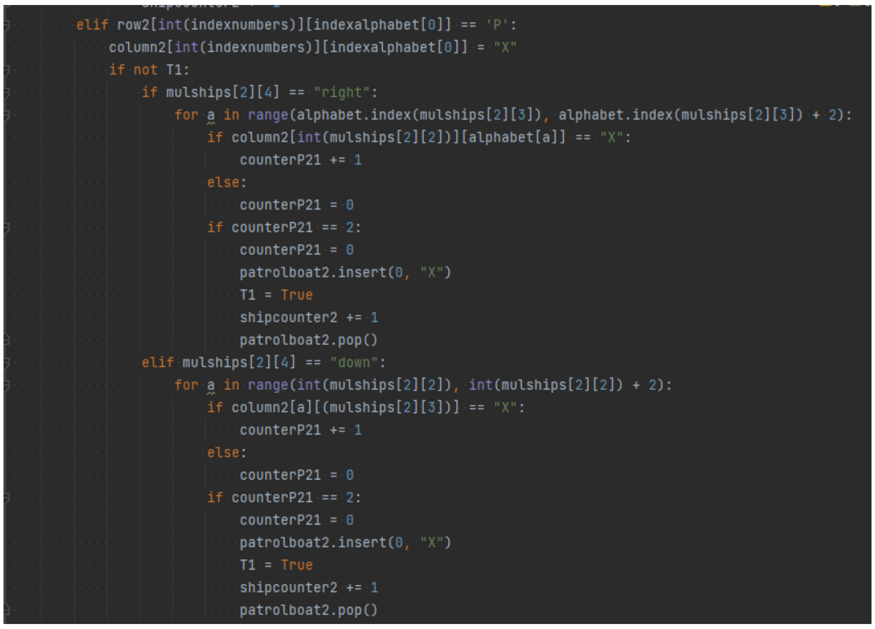
I assigned different counters for each ship to show the situation of the ship below the hidden board correctly.



```
carrier1, destroyer1, submarine1, destroyer2, submarine2, carrier2 = "-", "-", "-", "-", "-", "-"
patrolboat1, battleship1, patrolboat2, battleship2 = ["-", "-", "-", "-"], ["-", "-"], ["-", "-", "-", "-"], \
                                                     ["-", "-"]
counterS2, shipcounter2, shipcounter1, counterP14, counterP13, counterP12, counterP11, counterB12, counterB11, \
counterD2, counterC2, counterS1, counterB21, counterD1, counterC1, counterP24, counterP23, counterB22, counterP22,\
counterP21 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
(T1, T2, T3, T4, B1, B2, T5, T6, T7, T8, B3, B4) = (False, False, False, False, False, False, False, False,
False, False, False, False)
```

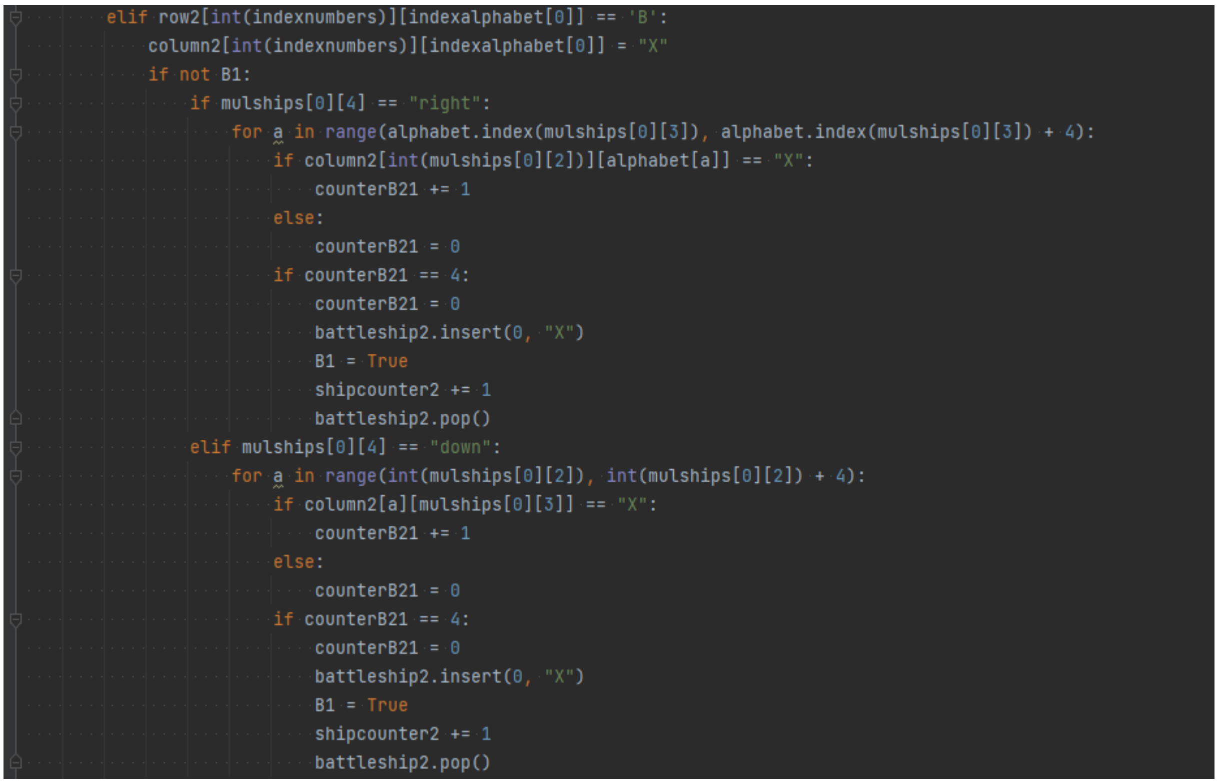
Ship name + number variables assigned to situations of ships. *Counter + first letter of ship + number* variables assigned to every part of the ships to check if the whole ship is sunk or not. *T + number* and *B + number* variables assigned to check how many multiple ships are sank or not.

This is one of the examples of how patrol boat ship sinking algorithm works. It counts if there is a *"P"* value at row and adds 1 to counter. If counter reaches to 2, checks it whether both 2 coordinates belong to the same ship. If it is, add one *"X"* element to patrol boat indicator *"patrolboat2"*.



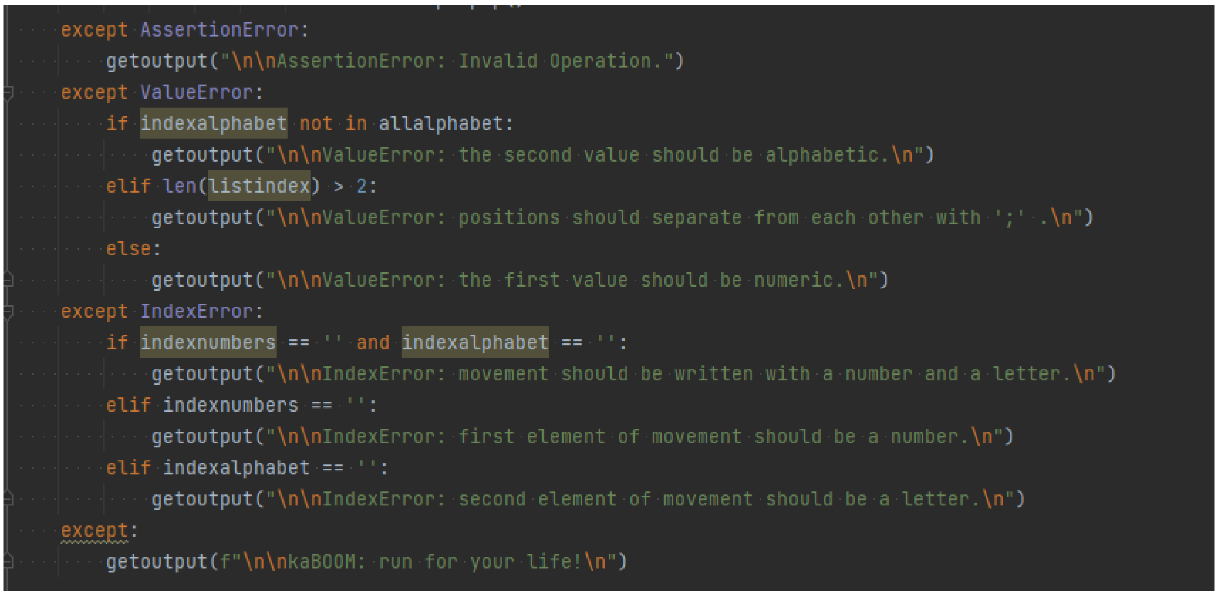
```
        elif row2[int(indexnumbers)][indexalphabet[0]] == 'P':
            column2[int(indexnumbers)][indexalphabet[0]] = "X"
            if not T1:
                if mulships[2][4] == "right":
                    for a in range(alphabet.index(mulships[2][3]), alphabet.index(mulships[2][3]) + 2):
                        if column2[int(mulships[2][2])][alphabet[a]] == "X":
                            counterP21 += 1
                        else:
                            counterP21 = 0
                        if counterP21 == 2:
                            counterP21 = 0
                            patrolboat2.insert(0, "X")
                            T1 = True
                            shipcounter2 += 1
                            patrolboat2.pop()
                elif mulships[2][4] == "down":
                    for a in range(int(mulships[2][2]), int(mulships[2][2]) + 2):
                        if column2[a][(mulships[2][3])] == "X":
                            counterP21 += 1
                        else:
                            counterP21 = 0
                        if counterP21 == 2:
                            counterP21 = 0
                            patrolboat2.insert(0, "X")
                            T1 = True
                            shipcounter2 += 1
                            patrolboat2.pop()
```

T1 variable checks if the first ship sunk. Through this, *"X"* counter works correctly.


```
        elif row2[int(indexnumbers)][indexalphabet[0]] == 'B':
            column2[int(indexnumbers)][indexalphabet[0]] = "X"
            if not B1:
                if mulships[0][4] == "right":
                    for a in range(alphabet.index(mulships[0][3]), alphabet.index(mulships[0][3]) + 4):
                        if column2[int(mulships[0][2])][alphabet[a]] == "X":
                            counterB21 += 1
                        else:
                            counterB21 = 0
                        if counterB21 == 4:
                            counterB21 = 0
                            battleship2.insert(0, "X")
                            B1 = True
                            shipcounter2 += 1
                            battleship2.pop()
                elif mulships[0][4] == "down":
                    for a in range(int(mulships[0][2]), int(mulships[0][2]) + 4):
                        if column2[a][mulships[0][3]] == "X":
                            counterB21 += 1
                        else:
                            counterB21 = 0
                        if counterB21 == 4:
                            counterB21 = 0
                            battleship2.insert(0, "X")
                            B1 = True
                            shipcounter2 += 1
                            battleship2.pop()
```

That is one of the examples of battleship algorithm from *play1*. It's the same base as patrol boat's. First, the value at the *column2* changes, then count of *"X"* changes in the indicator *"battleship2"*.

Because battleships and patrol boats multiple ships, I couldn't use a general loop for all of them. That's why my code has more than 500 row.


```
    except AssertionError:
        getoutput("\n\nAssertionError: Invalid Operation.")
    except ValueError:
        if indexalphabet not in allalphabet:
            getoutput("\n\nValueError: the second value should be alphabetic.\n")
        elif len(listindex) > 2:
            getoutput("\n\nValueError: positions should separate from each other with ';' .\n")
        else:
            getoutput("\n\nValueError: the first value should be numeric.\n")
    except IndexError:
        if indexnumbers == '' and indexalphabet == '':
            getoutput("\n\nIndexError: movement should be written with a number and a letter.\n")
        elif indexnumbers == '':
            getoutput("\n\nIndexError: first element of movement should be a number.\n")
        elif indexalphabet == '':
            getoutput("\n\nIndexError: second element of movement should be a letter.\n")
    except:
        getoutput(f"\n\nkaBOOM: run for your life!\n")
```

There are completely the same exceptions at the end of the *play1* and *play2* functions. They are the exception messages of the Errors which I introduced at the beginning of the function.

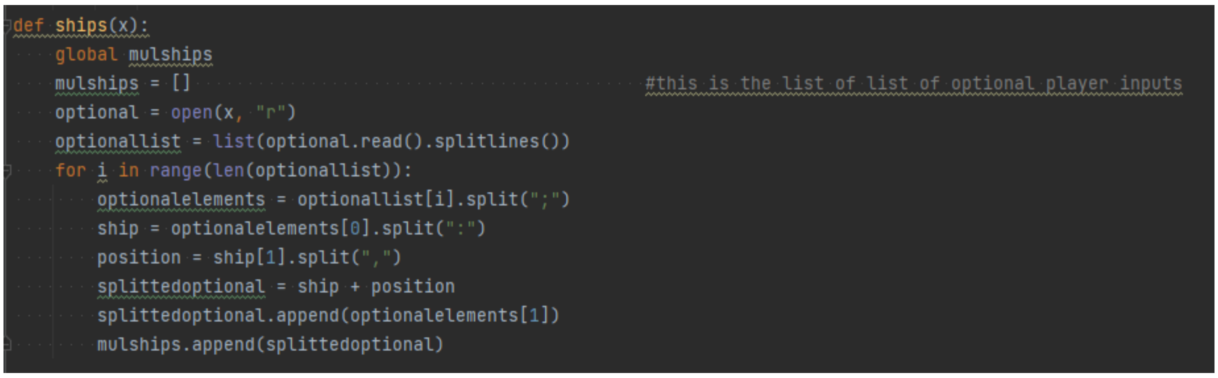
AssertionError occurs when inputs are correct form but out of number or alphabet range.

ValueError occurs when one of the elements of coordinate is given incorrectly or inputs separated with wrong signs.

IndexError occurs when given input has a missing element.

Function *ships():*

The variable *"mulships"* which is used in *play1* and *play2* functions frequently, created at *"ships"* function.

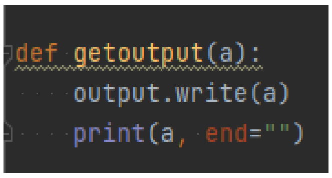


```
def ships(x):
    global mulships
    mulships = []                                          #this is the list of list of optional player inputs
    optional = open(x, "r")
    optionallist = list(optional.read().splitlines())
    for i in range(len(optionallist)):
        optionalelements = optionallist[i].split(";")
        ship = optionalelements[0].split(":")
        position = ship[1].split(",")
        splittedoptional = ship + position
        splittedoptional.append(optionalelements[1])
        mulships.append(splittedoptional)
```

This function just for reading and classifying the information in the *"OptionalPlayer1.txt"* and *"OptionalPlayer2.txt"* files. It reads the file and at the end of the function, form of "*mulships"* list become as:

[type and number of the ship, the origin point of ship, row number of origin point, column letter of origin point, which way the ship goes]

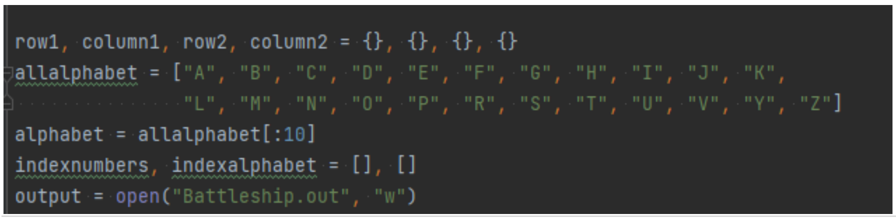
Function getoutput(a):



```
def getoutput(a):
    output.write(a)
    print(a, end="")
```

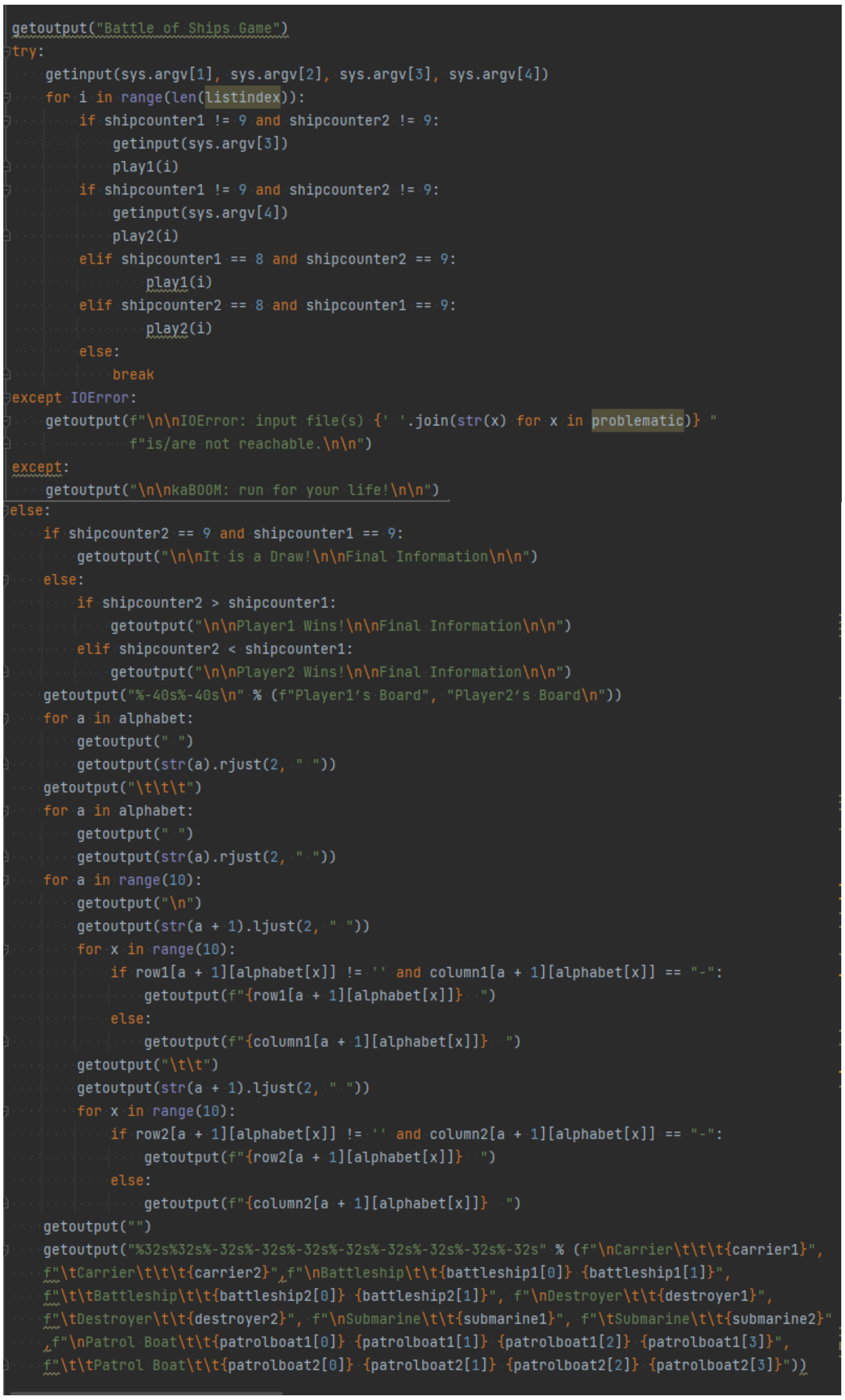
Basically, it writes the argument inside to output file *"Battleship.out"* and prints them to terminal at the same time.

(The beginning of the code, definition of output)



```
row1, column1, row2, column2 = {}, {}, {}, {}
allalphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K",
               "L", "M", "N", "O", "P", "R", "S", "T", "U", "V", "Y", "Z"]
alphabet = allalphabet[:10]
indexnumbers, indexalphabet = [], []
output = open("Battleship.out", "w")
```

The main code



```python
getoutput("Battle of Ships Game")
try:
    getinput(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
    for i in range(len(listindex)):
        if shipcounter1 != 9 and shipcounter2 != 9:
            getinput(sys.argv[3])
            play1(i)
        if shipcounter1 != 9 and shipcounter2 != 9:
            getinput(sys.argv[4])
            play2(i)
        elif shipcounter1 == 8 and shipcounter2 == 9:
                play1(i)
        elif shipcounter2 == 8 and shipcounter1 == 9:
                play2(i)
        else:
            break
except IOError:
    getoutput(f"\n\nIOError: input file(s) {' '.join(str(x) for x in problematic)} "
              f"is/are not reachable.\n\n")
except:
    getoutput("\n\nkaBOOM: run for your life!\n\n")
else:
    if shipcounter2 == 9 and shipcounter1 == 9:
        getoutput("\n\nIt is a Draw!\n\nFinal Information\n\n")
    else:
        if shipcounter2 > shipcounter1:
            getoutput("\n\nPlayer1 Wins!\n\nFinal Information\n\n")
        elif shipcounter2 < shipcounter1:
            getoutput("\n\nPlayer2 Wins!\n\nFinal Information\n\n")
    getoutput("%-40s%-40s\n" % (f"Player1’s Board", "Player2’s Board\n"))
    for a in alphabet:
        getoutput(" ")
        getoutput(str(a).rjust(2, " "))
    getoutput("\t\t\t")
    for a in alphabet:
        getoutput(" ")
        getoutput(str(a).rjust(2, " "))
    for a in range(10):
        getoutput("\n")
        getoutput(str(a + 1).ljust(2, " "))
        for x in range(10):
            if row1[a + 1][alphabet[x]] != '' and column1[a + 1][alphabet[x]] == "-":
                getoutput(f"{row1[a + 1][alphabet[x]]}  ")
            else:
                getoutput(f"{column1[a + 1][alphabet[x]]}  ")
        getoutput("\t\t")
        getoutput(str(a + 1).ljust(2, " "))
        for x in range(10):
            if row2[a + 1][alphabet[x]] != '' and column2[a + 1][alphabet[x]] == "-":
                getoutput(f"{row2[a + 1][alphabet[x]]}  ")
            else:
                getoutput(f"{column2[a + 1][alphabet[x]]}  ")
    getoutput("")
    getoutput("%32s%32s%-32s%-32s%-32s%-32s%-32s%-32s%-32s%-32s" % (f"\nCarrier\t\t\t{carrier1}",
    f"\tCarrier\t\t\t{carrier2}",f"\nBattleship\t\t{battleship1[0]} {battleship1[1]}",
    f"\t\tBattleship\t\t{battleship2[0]} {battleship2[1]}", f"\nDestroyer\t\t{destroyer1}",
    f"\tDestroyer\t\t{destroyer2}", f"\nSubmarine\t\t{submarine1}", f"\tSubmarine\t\t{submarine2}"
    ,f"\nPatrol Boat\t\t{patrolboat1[0]} {patrolboat1[1]} {patrolboat1[2]} {patrolboat1[3]}",
    f"\t\tPatrol Boat\t\t{patrolboat2[0]} {patrolboat2[1]} {patrolboat2[2]} {patrolboat2[3]}"))
```

To play the code, system arguments have taken in input function first. If there is a mistake, it will go one of the except columns.

Then, game will start at the last input file’s length (at this point, I consider both input files are in the same length.).

If a player finds all the ships of opponent, code checks whether the game is draw or not. If there is a possibility of draw, it will give one last chance to shot to the other player.

If a draw occurs, code goes to else and prints the draw boards. Otherwise, for loop will stop and code show the final situation, also the winner.

Pseudocode

1) the first day 10.12.22

I read the instruction and started to write the code. Firstly I created 4 functions:

getinput, getoutput, createboard and play.

probably i'll need more functions than i created.

```
getinput
try:
    if:
    read Player1's and Player2's board
    make a dictionary items of them
    read Player1's and Player2's inputs
    change the dict up to them
    else:
    create final board
except:
    error
```

That was the first skeleton of my code. The main functions are input and output functions. To make a file-readable code these functions are obligatory.

I decided to divide the code to functions. I used play function for each player and designed to take functions to a for loop. Through, game could play as long as input coordinates are given.

Considering try-except situations is pretty hard at first. I just created *IOError* and general except at first.

I decided to use dictionaries at input function because showing values of a dictionary and changing them with certain keys is way easier than changing a list’s. Creating the boards suitable with format was one of the hardest thing at the code. I thought that printing them like matrix in lists could work, and it did.

Algorithm just needs replace the dictionary values with given inputs. From this point, I started working based on errors and optional player files, not with an abstract plan.

Programmer’s Catalogue

As can be seen at pseudocode, I started to writing the code at 10.12.22. I didn’t work with a schedule all the time but I worked much. Probably this code took at least 40-50 hours to write in total. At first I tried to understand the assignment and wrote the basis. After I understand what should I do, planning and pseudocode part has begin.

I always did the testing at the same time as writing to make progress stable. I tested a draw ending, a winner player 1 ending and a winner player 2 winner ending on code. All of them worked correctly

Also I tested try-except situations after I finished writing. I wrote wrong input file names and incorrect input coordinates, changed the file names in path and my code worked stable.

I wrote my code reusable as soon as possible and made comments near codes to make it understandable to other programmers.

For example, *getinput()* function is able to take more than one arguments. It will give the correct outputs if the arguments is written in correct sequence.

Getoutput() function is able to write anything if it’s written in string format.

Play1() and *play2()* takes the input files from *getinput()* function, so the file changes does not affect them. Also, changing optional player files does not make any chance or cause any error. All of them is up to given inputs.

User's Catalogue

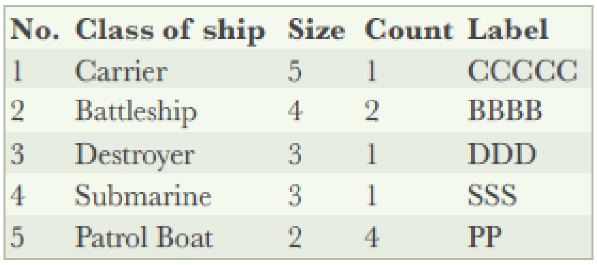
User's Manual (Tutorial)

Battleship is a game plays with 2 people. The aim is to sink opponent's ships while protect yours in a hidden board. To begin with, 6 file needed for this game.

1. Locations of player 1's ships
2. Locations of player 2's ships
3. Coordinates that player 1 want to shoot
4. Coordinates that player 2 want to shoot
5. The detailed locations of player 1's multiple ships
6. The detailed locations of player 2's multiple ships

Only the first four file should be given in this order (1-2-3-4) from the terminal. All the files and the game should be in the same path to play the game.

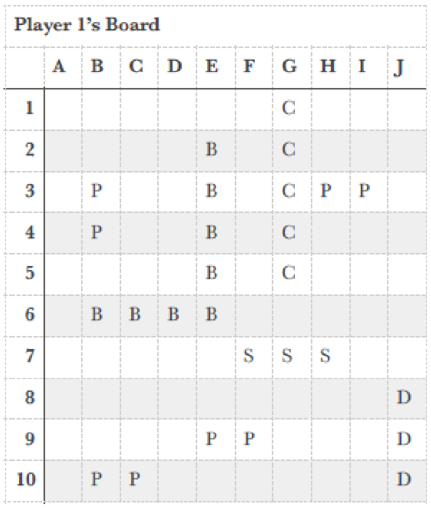
The game includes 5 types of game in different counts. Every player should locate their **all ships** at first.



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

There is table of ship types at the game. Every player should locate *1 carrier, 2 battleships, 1 destroyer, 1 submarine, 4 patrol boats.* The representation of them writes at *label* column. To place a ship, you should fill squares as size of ship. For example, if player 1 gives 3 "B" characters to game, one battleship of player 1 will not be counted.

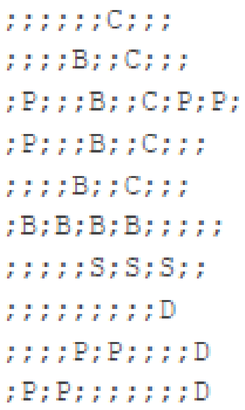
How to fill the files?



Player 1's Board

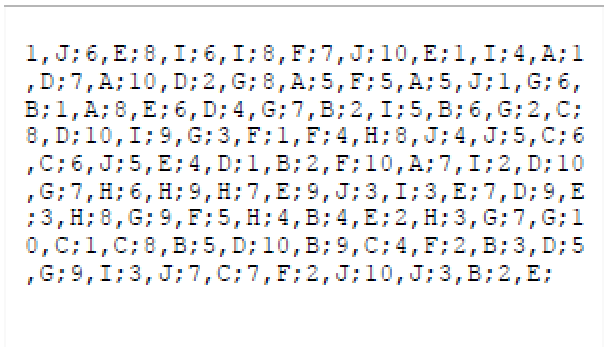
	A	B	C	D	E	F	G	H	I	J
1							C			
2					B		C			
3		P			B		C	P	P	
4		P			B		C			
5					B		C			
6		B	B	B	B					
7						S	S	S		
8										D
9					P	P				D
10		P	P							D

The location files should have ";" character for each grid and "C, B, P, S, D" characters for every part of the ship. Locations should be filled from left to right for every row. Here is an example of a correct location file at left. Locations should be written to a "txt" file for each player. Otherwise, game will give error.



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

The coordinate files should be filled with row numbers and column letters for every player. Coordinates which wanted to be shoot, should be given in “number,letter” format. Every coordinate should be split with “;” in files. Here’s an example of a correct coordinates input file.



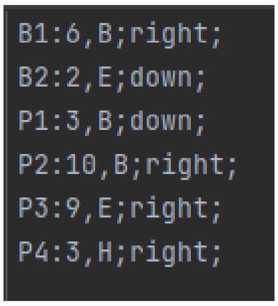
```
1,J;6,E;8,I;6,I;8,F;7,J;10,E;1,I;4,A;1
,D;7,A;10,D;2,G;8,A;5,F;5,A;5,J;1,G;6,
B;1,A;8,E;6,D;4,G;7,B;2,I;5,B;6,G;2,C;
8,D;10,I;9,G;3,F;1,F;4,H;8,J;4,J;5,C;6
,C;6,J;5,E;4,D;1,B;2,F;10,A;7,I;2,D;10
,G;7,H;6,H;9,H;7,E;9,J;3,I;3,E;7,D;9,E
;3,H;8,G;9,F;5,H;4,B;4,E;2,H;3,G;7,G;1
0,C;1,C;8,B;5,D;10,B;9,C;4,F;2,B;3,D;5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

Let’s consider this is player 1’s coordinate file. Game will shoot the coordinate row 1, column J at player 2’s hidden board in the first round.

Coordinate files should be “in” files otherwise, game will occur an error.

Also, there should be files for multiple ships for both players. Locations of patrol boat and battleship should be written in these files at this format:

[type and number of the ship, the origin point of ship, row number of origin point, column letter of origin point, which way the ship goes]



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

Here is an example for a correct multiple ships file for one player.

This file is automatically reading by *play1()* and *play2()* functions through *ships()* function. So, player do not have to take any action to read their file.

There are some functions that can be used to play the game.

play1():

This function first reads the coordinates file and the locations file of player 1. Given coordinate from player 1 reads and shoots in player 2’s board. If shot coordinate has a part of a ship, function will print “X” on player 2’s board. Otherwise, function will print “O” on player 2’s board.

play2():

this function does completely the same thing as play1 but for player 2. It reads player 2’s files and shoots player 1’s board.

Game ends if a player sank every ship of opponent. There should be 4 possibilities for endings:

1) Player 1 sank every ship of player 2 before them. Player 1 wins.
2) Player 2 sank every ship of player 1 before them. Player 2 wins.
3) Player 1 and 2 sank their ship at the same time. It’s a draw.
   a) Player 1 sank every ship of player 2 but player 1 just have one ship which did not sink. Last chance is given to player 2 and if the last ship of player 1, game ends draw.
   b) Player 2 sank every ship of player 1 but player 2 just have one ship which did not sink. Last chance is given to player 1 and if the last ship of player 2, game ends draw.
4) No player could sink every ship of opponent, game stops when the coordinates ended and nothing happens.

Restrictions

-Players must use “txt” files to write their ship’s locations to board.

-Locations must be written between “;” and with “C, B, P, D, S” letters (only uppercase).

-Players must use “in” files to write their coordinates they want to shoot.

-Coordinates should be written in given format below.

-Files for multiple ships are obligatory, they should be written in given format below.

-Only “1, 2, 3, 4, 5, 6, 7, 8, 9, 10” numbers and “A, B, C, D, E, F, G, H, I, J” letters can be used to write coordinates.

-A coordinate which has shot before cannot be given again.

-All ships players have must be placed.

-Every file which has given from system must be exist in the same path as the game.