

Assignment 4: Battle of Ships

<i>Name:</i> Yusuf İpek	<i>SN:</i> 222035048
<i>BBM103</i>	
Delivery Date: 03.01.2023	

Contents

Analysis	3
Design.....	3
Classes:.....	3
Lists:	3
Dictionaries:	3
Functions:	3
MainProgram.....	3
Get Arguments	4
Check Input	4
Read Input File	4
Create Output File	4
Write the Outputs	4
Split Start Positions	4
Split Moves	4
Check Moves.....	4
Game Round.....	4
Print Boards.....	4
Hit Boat	4
Game Over	4
Programmer's Catalogue.....	5
Imports	5
Defining Constants	5
Main Program.....	5
Get Arguments Funciton	6
File Functions.....	6
readInputFile Function	6
createOutputFile Function.....	6
writeFile Function	6
Splitting Inputs Functions	7
splitStartPositions Function.....	7
splitMoves Function.....	7
checkInputs Function	7
Player Class	8
Initialization and createPlayer Funciton.....	8

getBoatsIndex Function.....	8
checkBoats Function	8
groupingBoats Function	9
checkRowOrColumn Function.....	9
printBoards Function.....	10
printBoard Function	10
printBoats Function	10
gameRound Function	11
hitBoat Function.....	12
gameOver Function	12
checkMove Function	12
User Catalogue	13
File Hierarchy Examples	13
Txt Files Examples (Positions)	13
In Files Example (Moves).....	14
Outputs Example	14
Grading Table.....	15

Analysis

In the assignment, we have to write a battle ship game. The game gets input files name by command line arguments and reads positions and moves from the txt files which names are given. So user gives us to players ships positions and players moves in the beginning. We show the rounds and boards to user by looking given informations. Each round we show the players' moves and boards which shows the hitting positions. Also we show the count of the remaining ships. The game finishes when the at least one of the players all ships are hit or players moves are run out off. Also we have to handle exceptions.

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

Ship Types

In the game we have 5 different ship types. You can see in the “Ship Types” picture, some types of ships count is more than one so in the program we have to find the ships correctly.

Design

The program stores all player variables in player class, do writing and reading from files stuffs in functions. In the main function of program checks inputs, if there is something wrong, program quits itself and the game is not played. If everything is okey, the program starts to game and play rounds by looking players moves count. Every round 2 players make their moves and the program shows their moves in the hidden boards. The program using 2 list for that one of them is for hidden board which is shown just hit a boat or not with ‘X’ and ‘O’, the other one is for actual ships positions. The players can see just hidden boards until the end. Also program check the all ships of all players at the end of the each rounds. In the end of the game, players can see the remaining ships' positions in the boards.

Classes:

There is one class for player. In the Player Class, program stores name, positiones, moves, boards, boats variables. Also class has some methods, which is get boats index (by looking player positiones), grouping boats, check row or column (for grouping boats) and check boats (for end of the game).

Lists:

In the player class, the program has some list variables. The players'board, moves and positiones types are list. Also all of them are multi dimensional lists.

Dictionaries:

There is just one dictionary which is for ships. In this dictionary program has the informations of boat types. Also player has boats dictionary which is copy of the intial boats dictionary.

Functions:

MainProgram

Main function of the program. Check the errors or unexpected situtations in this function and play the game.

Get Arguments

Get command line arguments and check their count if it is unexpected write error message to output file and quit, else assign the file names to players.

Check Input

Check the input files by calling *read input file* function. Check all files, if there is an IOError, add the file names to wrong inputs variable and end of the checking, write the error message which shows the wrong input files names. If there is no any error return true.

Read Input File

Get the input file which is in the same directory of program. After that, read lines. For each line delete the end of the line string (\n) and add to a list. If there is any error return empty list.

Create Output File

Create a output file in the same directory. If the output file exists clear it.

Write the Outputs

Write the outputs line by line by appending txt file and printing console.

Split Start Positions

Split the data which is in input txt files by “;” and change the empty values with “-” and return multi-dimensional positions list.

Split Moves

Split the moves by looking “;” and return a moves list.

Check Moves

Check moves’ forms, valid form is “index,letter”. If there is something different, index is larger than 10 or letter is after than J, also the move is repeats itself write the proper error message and return false. If the form is valid return true.

Game Round

Write the general information (round count, grid size), players moves and call *print boards* function. Check the move indexes if the move is hitting any ships, call hit boat function and change the character of the board “-” to “X” otherwise if the move is not hitting any ships change the character with “O” and check all ships end of the round.

Print Boards

Print the players boards by looking final parameter. If final is true print the final informations boards, else print the hidden boats. Also write remaining boats’ counts.

Hit Boat

Remove the hitten place from list and calculate the count of the hitten ship.

Game Over

Write the winner or draw and write the final informations. (Boards with remaining ships places.)

Programmer's Catalogue

Imports

```
Assignment4.py > ...
1  from io import TextIOWrapper
2  import sys
3  import string
4  import copy
```

Import Modules Codes

Import the modules.

- IO for defining file variable which is for appending output file without open and close for each line.
- SYS for command line arguments
- STRING for getting alphabet list
- COPY for cloning a dictionary without copying its memory address just copy the values.

Defining Constants

```
outputFileName = "Battleship.out" # output file name
alphabet = list(string.ascii_uppercase) # alphabet list
initialBoats = {
    #keys are: first letters of {size, counts, names, indexes}
    'C': {'s': 5, 'c': 1, 'n': 'Carrier', 'i': []},
    'B': {'s': 4, 'c': 2, 'n': 'Battleship', 'i': []},
    'D': {'s': 3, 'c': 1, 'n': 'Destroyer', 'i': []},
    'S': {'s': 3, 'c': 1, 'n': 'Submarine', 'i': []},
    'P': {'s': 2, 'c': 4, 'n': 'Patrol Bot', 'i': []},
}
```

Constant Variables Codes

Define the constant variables,

- Alphabet List
- Output file name as "Battleship.out"
- Boats' informations: Names, counts, size and indexes

Main Program

```
def mainProgram():
    global outputFile
    createOutputFile()
    outputFile = open(outputFileName, "a") # open the output file in append mode
    if(getArguments() and checkInputs()):
        player1.createPlayer("Player1")
        player2.createPlayer("Player2")
        writeOutput("Battle of Ships Game\n")
        i = 0
        while(player1.index < len(player1.moves) and player2.index < len(player2.moves)):
            if(not gameRound(i)):
                return "Kaboom"
            i += 1
            player1.index += 1
            player2.index += 1
        writeOutput("No Winner !")
        printBoards(final=True)
        return "Kaboom"
    else:
        return "Kaboom !"
```

Main Program Codes

- Create output file if the file exist clear it with *createOutputFile* function.
- Define the output file and open it in append mode. (just beginning of the program)
- Check the arguments and inputs. If there is any problem return "kaboom" else keep going.
- Create players by *createPlayer* function of the *Player* class

- Write the output file name of the game.
- Check for the players' index. If it is smaller than their move count, call *gameRound* function with round count in a *while* loop, increment the indexes after *gameRound* function. Else, return "no winner" and write final informations.

Get Arguments Function

Return Type: *boolean*

```
def getArguments(): # get command line arguments
    global player1
    global player2
    try:
        player1_txt = sys.argv[1]
        player2_txt = sys.argv[2]
        player1_in = sys.argv[3]
        player2_in = sys.argv[4]
        player1 = Player(player1_txt, player1_in)
        player2 = Player(player2_txt, player2_in)
        return True
    except:
        writeOutput("command line arguments error !")
        return False
```

Get Arguments Function Codes

- Get the arguments from command line argument by using sys module.
- Initialize the players by their .txt and .in files.
- If there is any exception write output and return false, else return true.

File Functions

readInputFile Function

```
def readInputFile(fileName):
    dataList = []
    try:
        with open(fileName, "r") as _file:
            dataList = [line.replace("\n", "") for line in _file.readlines()]
    except IOError:
        return []
    return dataList
```

Read Input File Function Codes

Parameter: string *fileName*

Return Type: *List*

Get the input file with file name. Read all lines, remove end of the line character “\n” and add to the “dataList” list, return dataList. If there is any error return empty list.

createOutputFile Function

```
def createOutputFile():
    _file = open(outputFileName, "w")
    _file.write("")
    _file.close()
```

Create Output File Function Codes

If there is a output file which is named “Battleship.out” clear the file. Otherwise, create a file.

writeFile Function

```
outputFile: TextIOWrapper
def writeOutput(text, end = "\n"):
    outputFile.write(text + end) # append text to the file
    print(text, end=end) # write the command line
```

Write Output Function Codes

Parameters: string *text*, string *end*

Get the global output file variable (which is assigned in the *main program* function) and append the parameter *text* to into the file with adding *end* parameter to *text* parameter. Also print to the command line.

The *outputFile* defined global because of opening output file for every line is taking a bit long time

Splitting Inputs Functions

splitStartPositions Function

Parameter: List *dataList*

Return Type: List

```
def splitStartPosition(dataList):
    positions = []
    splittedData = [data.split(";") for data in dataList]
    linesPositions = []
    for data in splittedData:
        for _data in data:
            if(len(_data) == 0): linesPositions.append("-")
            else: linesPositions.append(_data)
        positions.append(linesPositions)
        linesPositions = []
    return positions
```

Split Start Positions Function Codes

Get the inputs with *readInputFile* function as a list.

Split the list by “;” character and change the empty characters with “-” character and add a new list for a line, add all lines to new list. At the end return positions multi-dimensional list which has the start positions in itself.

splitMoves Function

Parameter: List *dataList*

Return Type: List

```
def splitMoves(dataList):
    #moves are in just one line
    moves = dataList[0].split(";")[:-1] # delete last empty input
    if(moves == [""]): moves = []
    return moves
```

Split Moves Function Codes

Get the input data with *readInputFile* function as a list.

- Split it by looking “;” char and assign a list variable without last empty item and return moves list.

checkInputs Function

Return Type: boolean

```
def checkInputs(): # check for input files can readable
    inputs = [player1.position_txt, player1.moves_in, player2.position_txt, player2.moves_in]
    wrongInputs = ""
    for _input in inputs:
        if(not readInputFile(_input)):
            wrongInputs += _input + ", "
    if(len(wrongInputs) == 0):
        return True
    else:
        writeOutput(f"IOError: input file(s) {wrongInputs[:-2]} is/are not reachable.")
        return False
```

Check Inputs Function Codes

- Create a list which has the input files names.
- Define a *string* variable for wrong inputs.
- Call *readInput* function for each name in the inputs. If there is an error the *readInput* function return empty list and it is wrong in if statements. So if there is an error add the name of the wrong input to *string* variable.

- After looking for all items, check for the wrong inputs.
- If there is not any wrong input return *True*.
- Otherwise write the error message which has the wrong file names and return *False*.

Player Class

```
class Player:
    def __init__(self, position_txt, moves_in): # get input files name
        self.position_txt = position_txt
        self.moves_in = moves_in

    def createPlayer(self, name): # create player
        self.index = 0
        self.name = name
        self.positions = splitStartPosition(readInputFile(self.position_txt))
        self.moves = splitMoves(readInputFile(self.moves_in))
        self.board = [["-" for i in range(10)] for j in range(10)]
        self.boats = self.getBoatsIndex()
        self.groupingBoats()
```

Player Class Initialization and Create Player Function Codes

Initialization and createPlayer Function

Get the input files names in initialization.

Create the player:

- Get the inputs by reading input files and splitting their data (by *readInputFile* function and *split* functions),
- Define a empty board
- Get ships indexes and group them.
- Set the move index to 0.

getBoatsIndex Function

```
def getBoatsIndex(self):
    boats = copy.deepcopy(initialBoats) # just get initial values
    for i in range(10):
        for j in range(10):
            data = self.positions[i][j]
            if(data != "-"):
                boats[data]['i'].append(str(i)+str(j))
    return boats
```

Get Boats Index Function Codes

In the *Player* class,

- Copy the initial boats dictionary,
- By looking player's positions get the boats indexes and add a list for each boats,
- Return boats dictionary.

checkBoats Function

```
def checkBoats(self):
    # check for the remaining boats - game over control
    for boat in initialBoats:
        if(self.boats[boat]['c'] != 0):
            return True
    return False
```

Check Boats Function Codes

In the *Player* Class,

- In this function program checks the boats count if at least one of the boats have more than 0 component return *true* otherwise all boats get hit by other player and game is over so return *false*

groupingBoats Function

```
def groupingBoats(self): # grouping the boats/ get indexes and group them
    for boat in initialBoats:
        size = initialBoats[boat]['s']
        i = 0
        newIndexes = []
        while(i < len(self.boats[boat]['i'])):
            item = self.boats[boat]['i'][i]
            #check for same row
            row = self.checkRowOrColumn(self.boats[boat]['i'], int(item), size)
            if(row):
                newIndexes.append(row)
                i = 0
                continue
            #check for same column
            column = self.checkRowOrColumn(self.boats[boat]['i'], int(item), size, row=False)
            if(column):
                newIndexes.append(column)
                i = 0
            else:
                i += 1
        self.boats[boat]['i'] = newIndexes
```

Grouping Boats Function Codes

In the *Player* class,

- Check the indexes for each boats indexes in the initial boats dictionary.
- Check the indexes, if the indexes count is equal to the boats size add a new list and remove the indexes from dictionary.
- Check first row if there is not any ship check for the column.
- In the and add the new indexes list to the dictionary. So get the multi-dimensional indexes list in the boats dictionary.

checkRowOrColumn Function

```
def checkRowOrColumn(self,boatsIndexes:list, item:int, size:int, row = True):
    _range : range
    if(row):
        _range = range(item, item + size)
    else:
        _range = range(item, (item + size * 10), 10)
    for j in _range:
        j = str(j)
        if(len(j) < 2): j = "0" + j
        if(boatsIndexes.__contains__(j)):
            isFound = True
        else:
            isFound = False
            break
    if(isFound):
        newIndexes = []
        for i in _range:
            i = str(i)
            if(len(i) < 2): i = "0" + i
            newIndexes.append(str(i))
            boatsIndexes.remove(str(i))
        return newIndexes
    return []
```

Check Row Or Column Function Codes

Parameters: List *boatsIndexes*,
int *item*, int *size*, boolean *row*
Return Type: List

In the *Player* Class,

- Set the range by looking *row* parameter.
- If the parameter is true get the range by incrementing the index by 1,
- Else set the range for column, increment the count by 10 for new column.
- Check for the next squares until they are same and their size is equal to the parameter *size*. If squares are not same return empty list.
- If the program finds a ship add their indexes in a list and return it.

printBoards Function

```
def printBoards(final = False):
    def printBoard(player): ...
    def printBoats(): ...
    if(final):
        writeOutput("Final Informations\n")
        output = f"{player1.name}'s Board\t\t{player2.name}'s Board"
    else:
        output = f"{player1.name}'s Hidden Board\t\t{player2.name}'s Hidden Board"
    writeOutput(output)
    for i in range(11):
        printBoard(player1)
        writeOutput("\t", end="")
        printBoard(player2)
        writeOutput("")
    writeOutput("")
    printBoats()
```

Print Boards Function Codes

Parameter: boolean *final*

- Write the boards into the output file and console by writeOutput function.
- Check parameter final then write the boards.
- Call *printBoards* function for each player. Write the 2 tabs between the boards.
- Call *printBoats* function

```
def printBoard(player):
    board = player.board
    if(final):
        for boat in player.boats:
            dic = player.boats[boat]
            if(dic['c'] > 0):
                for index in dic['i']:
                    for inner in index:
                        player.board[int(inner[0])][int(inner[1])] = boat
    for j in range(11):
        if(i == 0 and j == 0):
            writeOutput(" ", end="")
            continue
        if(i == 0): #write Letters
            writeOutput(alpahet[j-1], end=" ")
        if(j == 0 and i>0): # write numbers
            writeOutput(str(i), end= " " * (2-len(str(i))))
        if(i > 0 and j > 0): # write indexes
            writeOutput(board[i-1][j-1],end=" ")
```

Print Board Function Codes

printBoard Function

In the *printBoards* function,

- Check for the final parameter.
- If it is true change the boats positions with their first letter if they are not get hit.
- Write the board (table) for given player. (the *end* parameter is "" because boards have to be side by side.)

```
def printBoats():
    players = [player1, player2]
    for boat in initialBoats:
        for i in range(2):
            playerBoatCount = players[i].boats[boat]['c']
            output = ""
            output += (initialBoats[boat]['c'] - playerBoatCount) * "X "
            output += playerBoatCount * "- "
            if(boat == "P"):
                tabCount = 2
            else: tabCount = 3
            writeOutput((initialBoats[boat]['n'] + "\t" + output + "\t" * tabCount), end = "")
        writeOutput("")
```

Print Boats Function Codes

printBoats Function

In the *printBoards* function,

- Get the all boats' counts.
- Write for the remainig boats "-" write for the hitten boats "X"
- If the boat is Petrol Boat use 2 tabs otherwise use 3 tabs between others players boats.

gameRound Function

Parameter: int *round*

Return Type: *boolean*

This function is main game function, the program calls this function for every round.

Parameter round is for showing round count.

```
def gameRound(round):
    players= [player1,player2]
    try:
        for i in range(len(players)):
            playerIndex = (i + 1) % 2 # other player
            writeOutput(f"{players[i].name}'s Move\n")
            writeOutput(f"Round : {round + 1}"+ "\t" * 5 + "Grid Size: 10x10\n")
            printBoards()
            moves = players[i].moves[players[i].index]
            writeOutput(f"\nEnter your move: {moves}")
            while( not checkMove(players[i])): # form is incorrect Look for correct move
                players[i].index += 1
                moves = players[i].moves[players[i].index]
                writeOutput(f"\nEnter your move: {moves}")
            move = moves.split(",")
            index1, index2=(int(move[0]) - 1, alphat.index(move[1]))
            indexStr = str(index1) + str(index2)
            data = players[playerIndex].positions[index1][index2]
            dataBoard = players[playerIndex].board[index1][index2]
            if(data == "-" and dataBoard == "-"):
                players[playerIndex].board[index1][index2] = "O"
            elif(data != "-" and dataBoard == "-"):
                if(players[playerIndex].boats[data]['c'] > 0):
                    for inner in players[playerIndex].boats[data]['i']:
                        if(inner.__contains__(indexStr)):
                            hitBoat(players[playerIndex].boats[data],inner, indexStr)
                            players[playerIndex].board[index1][index2] = "X"
            writeOutput("")
            isPlayer1Finish = not player1.checkBoats()
            isPlayer2Finish = not player2.checkBoats()
            if(isPlayer1Finish and isPlayer2Finish):
                gameOver(isDraw = True)
            elif(isPlayer1Finish):
                gameOver(player2.name)
            elif(isPlayer2Finish):
                gameOver(player1.name)
            else:
                return True
            return False # if game is over return false
    except IndexError:
        writeOutput("Index Error")
    except ValueError:
        writeOutput("Value Error")
    return False
```

Game Round Function Codes

- Create a list which has the players (player1 and player2).
- For every player, show their moves (for loop).
- Write general informations. (round count, grid size, ect.)
- Write boards with *printBoards* function
- Check for the move until the form of it is correct. By using *checkMove* function.
- Split the move to index and letter. Get letters index with using *alphabet* list.
- Check for the board and position list with given indexes.
- If the move is hitting a boat, call *hitBoat* function and change the character of the board given indexes element to "X".
- Else the position is empty so change it character to "O".

- After players' moves, check for the game over situation.
- If the game overs call the *gameOver* function with winner and return *False*. Also is there any error return *False* otherwise.
- If there is no any error and the game is not over return *True* and keep going to play.

hitBoat Function

```
def hitBoat(dic, indexes, item):
    indexes.remove(item)
    _count = 0
    for item in dic['i']:
        if (len(item) > 0):
            _count += 1
    dic['c'] = int(_count)
```

Hit Boat Function Codes

Parameters: dict *dic*, list *indexes*, string *item*

- Remove the item from the list.
- Calculate remaining boats' count for given boats. By calculating remaining positions of components of boats'.
- Update the count of it.

gameOver Function

```
def gameOver(playerName = "player", isDraw = False):
    if(isDraw):
        writeOutput("It is a Draw!")
    else:
        writeOutput(f"{playerName} Wins!\n")
    printBoards(final = True)
```

Game Over Function Codes

Parameters: string *playerName*, boolean *isDraw*

- Check for the *isDraw*.
- If it *isDraw* write the "draw"
- Else write the *playerName* and "Wins!"
- Call the *printBoards* function with parameter *final* is True For writing final informations.

checkMove Function

```
def checkMove(player):
    move = player.moves[player.index]
    splittedMove = move.split(",")
    try:
        if(len(splittedMove) < 2):
            raise IndexError # the expected input must have 2 items (e: 1,E) not less
        if(len(splittedMove) > 2):
            raise ValueError # the expected input must have 2 items (e 7,3) not more
        moveIndex = int(splittedMove[0]) #if the value is wrong this gives value error
        if(moveIndex > 10 or alphahet.index(splittedMove[1]) > 9):
            raise AssertionError
        if(player.moves[: player.index].__contains__(move)):
            raise AssertionError
        return True
    except IndexError:
        writeOutput(f"Index Error: Move Input {move} has less index than expected !")
    except ValueError:
        writeOutput(f"Value Error: Move Input {move} is wrong !")
    except AssertionError:
        writeOutput("AssertionError: Invalid Operation.")
    except:
        writeOutput("kaBOOM: run for your Life!")
    return False
```

Check Move Function Codes

Parameter: Player *player*

- Get the move by using player's moves list and player's index.
- Split the move by "," looking char.
- Check for the length if it is less than expected raise index error, if it is greater the expected raise value error.
- If move's first item is not int raise value error
- If the move's items is not in the table or the move repeats itself raise assertion error.
- If there is not any error return *True*. Else write the error message and return *False*.

User Catalogue

1. Write the boats positions for each player to txt file.
2. Write the players moves to in file.
3. Execute the python file (Assignment4.py) with input files names. First two txt files are for players' ships' positions, other input files are for players' moves.

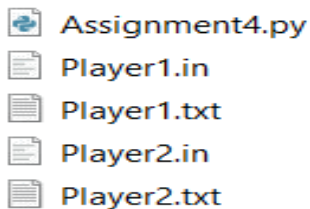
(**Example Command:** `python3 Assignment4.py "Player1.txt" "Player2.txt" "Player1.in" "Player2.in"`)

```
python Assignment4.py "Player1.txt" "Player2.txt" "Player1.in" "Player2.in"
```

Execution Command Example

4. Python file writes the outputs to Battleship.out file and command line.
5. Check out the output file (Battleship.out) and command line.

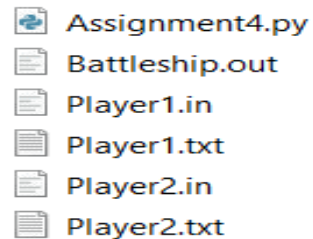
File Hierarchy Examples



Assignment4.py
Player1.in
Player1.txt
Player2.in
Player2.txt

File Hierarchy Before Execution

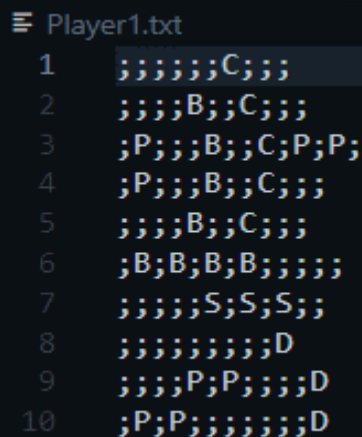
The program creates
"Battleship.out" file
when the user send
execution
command.



Assignment4.py
Battleship.out
Player1.in
Player1.txt
Player2.in
Player2.txt

File Hierarchy After Execution

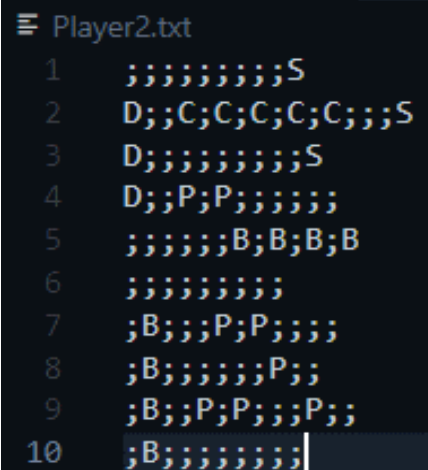
Txt Files Examples (Positions)



```
Player1.txt
1  ;;;;;;C;;;
2  ;;;;B;;C;;;
3  ;P;;;B;;C;P;P;
4  ;P;;;B;;C;;;
5  ;;;;B;;C;;;
6  ;B;B;B;B;B;
7  ;;;;S;S;S;
8  ;;;;;;;;;;D
9  ;;;;P;P;D
10 ;P;P;D
```

*Players Ships Positions Example
(Player1.txt)*

The grid size is 10x10. You can see 2 different positions txt file in the examples. Each player write their ships' first letter for specify their ships' positions. In the program the ships get grouped by looking row and columns. You have to write the positions with no confusion.



```
Player2.txt
1  ;;;;;;;;;;S
2  D;;C;C;C;C;C;S
3  D;;;;;;;;;S
4  D;;P;P;
5  ;;;;B;B;B;B
6  ;;;;;;;;;;
7  ;B;;P;P;
8  ;B;;P;
9  ;B;;P;P;P;
10 ;B;;P;P;P;P;
```

*Players Ships Positions Example
(Player2.txt)*

PPP
P

There is just one petrol boat for the program. So there is a confusion.

If you write like right picture, the program get two ships one is top one is bottom but if you write 3P-1P like left picture, the program get just one ship which is start from top left and have just 2 indexes. So be careful to writing starting ships' positons !

PP
PP

There is two petrol boats. There is not any confusion for the program.

In Files Example (Moves)

```
Player2.in
Player2.in
1 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
```

Input File Example (player2's moves)

You have to write moves in one line. You should write the moves "index,letter" form and put ";" between each moves.

If you don't write moves like example program can not accept the moves and writes the "wrong input" in their round, after that goes to the next move until the move form is going to be correct.

```
Enter your move: 11,A
AssertionError: Invalid Operation
Enter your move: 5,E
```

Wrong Input Example

If the form is not index,letter program write value error, if the move has just one letter or integer program write index error. The move have more than one move like ;2,A,B; this cause a value error (when the move components is more than 2). Also you should write less than 11 for integer and write a letter which is before than J (j is included), if you don't the program writes assertion error. And move repetition causes the assertion error too.

which is before than J (j is included), if you don't the program writes assertion error. And move repetition causes the assertion error too.

Outputs Example

```
Konsole
Komut yazın
kullanıcı etkileşime ya da veri alınması gerektiren komutları yürütülmesin
Geriçi kladır
./mtd/dad1/ogrenci/ogr/52220356046/BDG103/Assignments/assignment 4/assignment4-52220356046

3 - X - 0 X 0 X X 0
4 0 X - 0 X 0 X 0 - 0
5 0 0 0 0 X 0 X 0 - 0
6 - X X X X - 0 0 0 0
7 0 0 0 0 X X X 0 0
8 0 0 - 0 0 0 0 - 0 X
9 - - 0 - X X 0 0 X
100 X X 0 0 - 0 - 0 X

Carrier X
Battleship X
Destroyer X
Submarine X
Patrol Bot X X X X
Enter your move : 2,E

Player2 Wins!

Final Informations
Player1's Board
A B C D E F G H I J
1 0 0 0 0 - 0 X - 0 0
2 - 0 0 0 X 0 X 0 0 0
3 - X - 0 X 0 X X 0 - 0
4 0 X - 0 X 0 X 0 - 0
5 0 0 0 0 X 0 X 0 - 0
6 - X X X X - 0 0 0 0
7 0 0 0 0 X X X 0 0
8 0 0 - 0 0 0 0 - 0 X
9 - - 0 - X X 0 0 X
100 X X 0 0 - 0 - 0 X

Carrier X
Battleship X X
Destroyer X
Submarine X
Patrol Bot X X X X

Player2's Board
A B C D E F G H I J
1 - 0 0 0 0 0 - 0 X
2 0 0 X X X X X 0 0 X
3 X 0 0 0 0 0 0 0 - X
4 X 0 X X 0 0 0 0 0 0
5 - - - - 0 0 X X X X
6 0 0 0 0 - - 0 0 0 -
7 0 X 0 0 X X X 0 0 0
8 0 X - 0 0 0 0 X 0 0
9 - X 0 X X 0 - X 0 -
100 X 0 0 0 0 0 0 0 0

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

Console Output Example

```
Battle of Ships Game
Player1's Move

Round :1
Grid Size: 10x10

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

Carrier -
Battleship -
Destroyer -
Submarine -
Patrol Bot - - - -
Enter your move : 5,E

Player2's Move

Round :1
Grid Size: 10x10

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

Carrier -
Battleship -
Destroyer -
Submarine -
Patrol Bot - - - -
```

Output File Example

It is a Draw!
Final Informations

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 0 0 0 0 - 0 X - 0 0	1 - - 0 - - 0 0 - 0 X
2 - 0 0 0 X 0 X 0 0 0	2 X 0 X X X X X 0 0 X
3 - X - 0 X 0 X X X 0	3 X 0 0 0 0 0 0 0 - X
4 0 X - 0 X 0 X 0 - 0	4 X 0 X X 0 0 0 0 0 0
5 0 0 0 0 X 0 X 0 - 0	5 - - - - 0 0 X X X X
6 - X X X X - 0 0 0 0	6 0 0 0 0 - - 0 0 0 -
7 0 0 0 0 X X X 0 0	7 0 X 0 0 X X 0 0 0 0
8 0 0 - 0 0 0 0 - 0 X	8 0 X - 0 0 0 0 X 0 0
9 - - 0 - X X 0 0 X	9 - X 0 X X 0 - X 0 -
100 X X 0 0 - 0 - 0 X	100 X 0 0 0 0 0 0 0 0
Carrier X	Carrier X
Battleship X X	Battleship X X
Destroyer X	Destroyer X
Submarine X	Submarine X
Patrol Bot X X X X	Patrol Bot X X X X

Output File Example (For Draw)

Grading Table

Evaluation	Points	Evaluate Myself (Guess Grading)
Readable Codes and Meaningful Naming	5	5
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	5
Function Usage	15	15
Correctness, File I/O	30	28
Exceptions	20	20
Report	20	20
There are several negative evaluations	...	-2
Total	100	96