BBM 103:

INTRODUCTION TO PROGRAMMING LABORATORY I

FALL 2022

Assignment 4: Battle of Ships

2210356056
Sinan DOĞAN

23.12.2022

Table of Contents

Analysis

Battleship is a game for two players. It is played on ruled grids on which each player's fleet of warships is marked. The locations of the fleets are concealed from the other player. Players alternate turns calling shots at the other player's ships, and the game's objective is to destroy the opposing player's fleet.

The game is played in four grids of squares 10x10, two for each player. On one grid, the player arranges ships and records the shots by the opponent. On the other grid, the player records their shots. Before play begins, each player secretly arranges the ships on their hidden grid. Each ship occupies several consecutive squares on the grid, arranged either horizontally or vertically. The type of ship determines the number of squares for each ship. The game is a discovery game in which players must discover their opponents' positions.

Design

The boards of the players will be created by reading the one for the board from the input files received from the user. After the boards are created, the game will be played with the moves in the other input files. An error message will be given to the user if given input file name is wrong. In case of incorrectly entered moves (IndexError, ValueError), an error message will be given and the same player will continue to play with their next move.

All output will be printed both to the terminal and to battleship.out file.

- writeOutput, writeColumn, columnAppend:

```
append each output in rightColumn list or leftColumn list according to which
column it will be in
print an element firstly from the leftColumn list, then from the rightColumn list
write an element firstly from the leftColumn list, then from the rightColumn list
```

These functions print the output to be given in order by appending it in the list according to the column to be written.

- boardCreator:

```
read the input file for each player
create a board for each player according to the input file
create a hidden board for each player according to the input file
place ships according to the input file to the board
```

This function reads the input file and creates a normal board and a hidden board for the player.

- game:

```
make moves according to the player's input file
if there is an error
    print error message
    try again with next move
else
    if there is a ship at the hit location
        write X in that location on the player’s board
    else
        write O in that location on the player’s board
```

The player makes his move according to the number of turns. If the entered move is incorrect, an error message is printed and the next move is made. If there is a ship in the location where the move is made, print “X” on the board, otherwise print “O”.

- showGame:

```
if playerhidden
    print playerhidden board
else
    print player board
print ships and information on how many ships are left
```

This function prints the player's board and information about how many ships are left.

- shipFunction, checkRow, checkColumn:

```
append location of all ships to shipComplex list
check the row and column
group ships according to positions the ship
```

This function detects and groups the player's ship parts.

- counter, countShips:

```
return for each ship how many ships are on the board based on the given ship
letter
```

These functions provide information about the player's unsinkable ship information.

- openFile:

```
try to open the given file
    if the file is cannot open
        append the file name in to the errorFiles list
    else
        open the file
```

This function tries to open the given file. If it can't, it appends the file name to errorFiles list.

Programmer's Catalogue

Lists & Dictionaries

```
leftColumn = list()
rightColumn = list()
board = {"Player1": {}, "Player2": {}, "Player1hidden": {}, "Player2hidden": {}}
shipsComplex = {"Player1": {}, "Player2": {}}
shipsCategorized = {"Player1": {}, "Player2": {}}
errorFiles = list()
moves = {"Player1":[],"Player2":[]}
```

leftColumn list contains the texts that will be written in the left column in the output.

rightColumn list contains the texts that will be written in the right column in the output.

board dictionary contains the boards and normal boards of the players.

shipsComplex dictionary contains a list of the coordinates of each player's ships.

```
example of shipsComplex for Player1:
All ships are in this dictionary, but not categorized
```
```
{(1, 6): 'C', (2, 4): 'B', (2, 6): 'C', (3, 1): 'P', (3, 4): 'B', (3, 6): 'C', (3, 7): 'P', (3, 8): 'P', (4, 1): 'P', (4, 4): 'B', (4, 6): 'C', (5, 4): 'B', (5, 6): 'C', (6, 1): 'B', (6, 2): 'B', (6, 3): 'B', (6, 4): 'B', (7, 5): 'S', (7, 6): 'S', (7, 7): 'S', (8, 9): 'D', (9, 4): 'P', (9, 5): 'P', (9, 9): 'D', (10, 1): 'P', (10, 2): 'P', (10, 9): 'D'}
```

shipsCategorized dictionary contains a grouped list of each player's ships.

```
example of shipsCategorized for Player1:
Ships are categorized according to the class
```
```
{'C': {1: [(1, 6), (2, 6), (3, 6), (4, 6), (5, 6)]}, 'B': {1: [(2, 4), (3, 4), (4, 4), (5, 4)], 2: [(6, 1), (6, 2), (6, 3), (6, 4)]}, 'D': {1: [(8, 9), (9, 9), (10, 9)]}, 'S': {1: [(7, 5), (7, 6), (7, 7)]}, 'P': {1: [(3, 1), (4, 1)], 2: [(3, 7), (3, 8)], 3: [(9, 4), (9, 5)], 4: [(10, 1), (10, 2)]}}
```

errorFiles list contains the names of the files that could not be opened.

moves dictonary contains each player's moves in the input file.

Output Functions: writeOutput(write), writeColumn(left,right), columnAppend(first,second)

```
def writeOutput(write):
    print(write, end="")
    file.write(write)

def writeColumn(left,right):
    for first,second in zip(left,right):
        if second == "":
            writeOutput(f"{first}\n")
        else:
            lenFirst = len(first.expandtabs(4))
            space = ((28 - lenFirst)//4) if (lenFirst % 4) == 0 else ((28 - lenFirst)//4 + 1)
            tab = "\t" * space
            writeOutput(f"{first}{tab}{second}\n")

def columnAppend(first,second=""):
    if type(first) is str:
        leftColumn.append(first)
        rightColumn.append(second)
    else:
        leftColumn.extend(first)
        rightColumn.extend(second)
    leftColumn.append("")
    rightColumn.append("")
```

writeOutput function prints to the terminal and writes to the battleship.out file.

writeColumn function prints the outputs left and right column. It zips left and right column with for loop. If rightColumn is empty, it prints only lefColumn. If not, it calculate how many space have to be between left and right column, and prints left column, space between two columns and right column.

columnAppend function adds outputs to the leftColumn and rightColumn. If given output is a string, it appends with append function, if given output is a list, it adds with extend funciton. After all that, it appends a blank line to leftColumn and rightColumn.

Board Functions: boardCreator(player,file)

```
def boardCreator(player, file): # to create a board and a hidden board
    board[player][" "] = tuple(ascii_uppercase[:10]) # alphabet line
    board[player].update({i: list("-"*10) for i in range(1,11)})
    for number,line in enumerate(file.readlines(), start=1):
        line = line.rstrip("\n").split(";")
        column = 0

        for letter in line:
            if letter != "": # if letter is not empty, that show us there is a ship
                board[player][number][column] = letter # the position is changed
            column += 1

    board[player+"hidden"][" "] = tuple(ascii_uppercase[:10])
    board[player+"hidden"].update({i: list("-"*10) for i in range(1,11)})
```

boardCreator function's parametres are player and file. Firstly, it creates a board and a hidden board for player. Then, it reads the input file of player, splits input file by ";" and if there is a ship in the location, function places the ship on the location in the board.

Ship Functions: shipFunction(player), checkRow(**), checkColumn(**), counter(player, letter), countShips(player)

```
def shipFunction(player):
    for row, line in enumerate(list(board[player].values())[1:], start=1):
        for column, ship in enumerate(line):
            if ship != "-":
                shipsComplex[player][row,column] = ship

    for shipLetter in ("C","B","D","S","P"):
        number = 1
        if shipLetter == "C":
            howManyShips = 5
        elif shipLetter == "B":
            howManyShips = 4
        elif shipLetter == "D" or shipLetter == "S":
            howManyShips = 3
        elif shipLetter == "P":
            howManyShips = 2

        for ship in shipsComplex[player]:
            if shipsComplex[player][ship] == shipLetter:
                if checkRow(player,ship,howManyShips,howManyShips,number,shipLetter):
                    number += 1
                elif checkColumn(player,ship,howManyShips,howManyShips,number,shipLetter):
                    number += 1
```

shipFunction function adds the player's ships uncategorized to shipsComplex dictionary. Then, all ships in shipsComplex dictionary are categorized by checkRow and checkColumn functions.

```python
def checkRow(player,ship,times,shipsCount,number,shipLetter):
    try:
        if shipLetter == shipsComplex[player][(ship[0],ship[1]+times)]:
            return checkColumn(player,ship,times,shipsCount,number,shipLetter)
    except:
        pass
    while times > 0:
        times -= 1
        if ship in shipsComplex[player]:
            return checkRow(player,(ship[0],ship[1]+1),times,shipsCount,number,shipLetter)
        else:
            return False

    while shipsCount > 0:
        if shipLetter not in shipsCategorized[player]:
            shipsCategorized[player][shipLetter] = dict()
            shipsCategorized[player][shipLetter][number] = [(ship[0],ship[1]-shipsCount)]
        else:
            if number not in shipsCategorized[player][shipLetter]:
                shipsCategorized[player][shipLetter][number] = list()
            shipsCategorized[player][shipLetter][number].append((ship[0],ship[1]-shipsCount))
        shipsComplex[player][(ship[0],ship[1]-shipsCount)] = "X"
        shipsCount -=1
    return True

def checkColumn(player,ship,times,shipsCount,number,shipLetter):
    while times > 0:
        times -= 1
        if ship in shipsComplex[player]:
            return checkColumn(player,(ship[0]+1,ship[1]),times,shipsCount,number,shipLetter)
        else:
            return False

    while shipsCount > 0:
        if shipLetter not in shipsCategorized[player]:
            shipsCategorized[player][shipLetter] = dict()
            shipsCategorized[player][shipLetter][number] = [(ship[0]-shipsCount,ship[1])]
        else:
            if number not in shipsCategorized[player][shipLetter]:
                shipsCategorized[player][shipLetter][number] = list()
            shipsCategorized[player][shipLetter][number].append((ship[0]-shipsCount,ship[1]))
        shipsComplex[player][(ship[0]-shipsCount),ship[1]] = "X"
        shipsCount -=1
    return True
```

checkRow and checkColumn functions are have same logic. These functions group ships by controlling the row and column of each letter. If it is found to be a ship, it appends their location to the shipsCategorized dictionary.

```python
def counter(player, letter):
    result = 0
    for ship in shipsCategorized[player][letter].values():
        if ship != []:
            result += 1

    return result

def countShips(player):
    carrier = counter(player, "C")
    battleship = counter(player, "B")
    destroyer = counter(player, "D")
    submarine = counter(player, "S")
    patrolboat = counter(player, "P")

    return carrier,battleship,destroyer,submarine,patrolboat
```

counter function counts and returns the ships whose letter is given on the board of player.

countShips function returns the number of ships that have not yet sunk for each ship.

Game Functions: game(player,otherPlayer,round), showGame(player,hidden)

```
def game(player,otherPlayer,round):
    columnAppend(f"{player}'s Move")
    columnAppend(f"Round : {round}","Grid Size: 10x10")
    while len(moves[player]) >= round:
        try:
            move = moves[player][round-1]
            if "," not in move:
                raise IndexError
            number, letter = move.split(",")
            if number == "" or letter == "":
                raise IndexError
            number, letter = int(number), str(letter)
            column = ascii_uppercase.index(letter)
            assert 0 < number <= 10
            assert letter in ascii_uppercase[0:10]
        except IndexError:
            columnAppend(f"IndexError: The given input is incorrect.")
            round += 1
        except ValueError:
            columnAppend(f"ValueError: The given input is incorrect.")
            round += 1
        except AssertionError:
            columnAppend("AssertionError: Invalid Operation")
            round += 1
        else:
            columnAppend(showGame("Player1"),showGame("Player2"))
            columnAppend(f"Enter your move: {move}")
            number_letter = board[otherPlayer][number][column]

            if number_letter != "-":
                classOfShipHit = board[otherPlayer][number][column]
                for ships in shipsCategorized[otherPlayer][classOfShipHit].values():
                    if (number,column) in ships:
                        ships.remove((number,column))
                board[otherPlayer+"hidden"][number][column] = "X"
                board[otherPlayer][number][column] = "X"
            else:
                board[otherPlayer+"hidden"][number][column] = "O"
                board[otherPlayer][number][column] = "O"

            break
```

game function allows players to do their moves. It makes the player move according to the number of rounds. If the given move is incorrect, the error message is printed and the next move is made. If there is a ship in the position where the player made a move, that position is marked with an "X" letter on the other player's board and hidden board. if no ship is marked with an "O" letter.

```
def showGame(player, hidden=True):
    output = list()
    if hidden:
        output.append(f"{player}'s Hidden Board")
        for i in board[player+'hidden']:
            output.append(f"{i:<2}{' '.join(board[player+'hidden'][i])}")
    else:
        output.append(f"{player}'s Board")
        for i in board[player]:
            output.append(f"{i:<2}{' '.join(board[player][i])}")

    carrier,battleship,destroyer,submarine,patrolboat = countShips(player)
    output.append("")
    output.append(f"Carrier\t\t{' '.join('X'*(1-carrier) + '-'*(carrier))}")
    output.append(f"Battleship\t{' '.join('X'*(2-battleship) + '-'*(battleship))}")
    output.append(f"Destroyer\t{' '.join('X'*(1-destroyer) + '-'*(destroyer))}")
    output.append(f"Submarine\t{' '.join('X'*(1-submarine) + '-'*(submarine))}")
    output.append(f"Patrol Boat\t{' '.join('X'*(4-patrolboat) + '-'*(patrolboat))}")

    return output
```

showGame function gives the current state of the game, prints the boards and provides the remaining ship information using the countShips function and creates a ship table with this information.

File Function: openFile(number)

```
def openFile(number):
    try:
        return open(sys.argv[number],"r")
    except:
        errorFiles.append(sys.argv[number])
```

fileFunciton tries to open the file located at the given number (sys.argv[number]). If it encounters an error, it adds the filename to the errorFiles list. Thus, if one or more files are not exist in the directory, file name/s can be printed in error messages.

Game:

```
try:
    file = open("Battleship.out", "w")
    errorFiles = list()
    player1txt, player2txt = openFile(1), openFile(2)
    player1in, player2in = openFile(3), openFile(4)
    if len(errorFiles) != 0:
         raise IOError
    moves = {"Player1":[],"Player2":[]}
    moves["Player1"] = player1in.read().rstrip(";").split(";")
    moves["Player2"] = player2in.read().rstrip(";").split(";")
    boardCreator("Player1",player1txt)
    boardCreator("Player2",player2txt)
    shipFunction("Player1")
    shipFunction("Player2")
    columnAppend(f"Battle of Ships Game")
    round = 1
    while len(moves["Player1"]) >= round and len(moves["Player2"]) >= round:
        game("Player1","Player2",round)
        game("Player2","Player1",round)
        round += 1

    if sum(countShips("Player1")) == 0:
        columnAppend("Player2 Wins!")
    elif sum(countShips("Player2")) == 0:
        columnAppend("Player1 Wins!")
    elif sum(countShips("Player1")) == 0 and sum(countShips("Player2")) == 0:
        columnAppend("It is a Draw!")
    else:
        columnAppend("Something went wrong, no winner.")

    columnAppend("Final Information")
    columnAppend(showGame("Player1",False),showGame("Player2",False))
except IOError:
    columnAppend(f"IOError: input file(s) {', '.join(errorFiles)} is/are not reachable.")
except IndexError:
    columnAppend("IndexError: number of input files less than expected.")
except:
    columnAppend("kaBOOM: run for your life!")
else:
    player1txt.close()
    player2txt.close()
    player1in.close()
    player2in.close()
finally:
    writeColumn(leftColumn,rightColumn)
    file.close()
```

The codes above are where the game happens. All defined functions are used here.

Output file is opened, input file is tried to be opened. If one or more input files are incorrect, an error message is given.

According to the input file, moves are added to the moves dictionary.

Boards are created for both players and their ships are detected.

The 1st round, starts, the while loop repeats until the number of moves of the players ends. First player 1, then player 2 performs the step and the number of rounds is increased by 1.

When the game is over, if the 1st player's remaining number of ships is 0, the 2nd player wins the game, and if the 2nd player's remaining number of ships is 0, the 1st game wins the game. If the remaining number of ships in both of players is 0, the game end in a draw. If otherwise, something went wrong with the code (the moves given may be insufficient to finish the game). Final information is given and the non-hidden board of the players is printed.

If there is any unexpected error "kaBOOM: run for your life!" is printed. If there is no error, opened input files are closed. In any case, the output file is printed (even in case of an error, the output file is printed) and closed.

User Catalogue

Program's User Manual & Tutorial

- Input Files: (Player1.txt, Player2.txt)

In the Player1.txt and Player2.txt file, the player needs to place their ships. **1**, size 5 **Carrier**; **2**, size 4 **Battleships**; **1**, size 3 **Destroyer**; **1** size 3 **Submarine** and **4**, size 2 Patrol **Boats** will be positioned.

A semicolon ";" will be placed at the end of each column.

Example: Input:

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

	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

- Input Files: (Player1.in, Player2.in)

The player will input his/her moves into the Player1.in and Player2.in files. Each move will be separated by a semicolon ";". The row and column will be separated by comma ",", the left side of the comma will give the row number and the right side will give the column letter.

Example: Input:

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

Output File: (battleship.out)

When the program is run, a file named battleship.out will be created in the same directory and the information will be written into that file and to the terminal. Output file will write which player's move, number of rounds, grid size, players' hidden boards, ship information and information of the move to be made. At the end of the output file, there will be the final information.

Examples: a Part Of The Output

```
Player1's Move

Round : 64                  Grid Size: 10x10

Player1's Hidden Board      Player2's Hidden Board
  A B C D E F G H I J         A B C D E F G H I J
1 O O - O - O X - O O       1 - - - - - - O - - -
2 - - O O - O X O O -       2 - - X X X X X O O X
3 - - - - X O - X X -       3 - O O O O O O - - X
4 O X - O X - X O - O       4 X O X - O O O O O O
5 O O O - X O - O - O       5 - - - - O O X X - X
6 - X X X X - O O O O       6 O O O - - - O O - -
7 O O - O O - - X O O       7 O - - O - X - O O -
8 O - - O O O O - O X       8 O - - O O O O X O -
9 - - - - X X O O - X       9 - X O X X O - X O -
100 - - O O - O - O -       100 X O O O - O O O O

Carrier      -              Carrier      X
Battleship   X -            Battleship   - -
Destroyer    -              Destroyer    -
Submarine    -              Submarine    -
Patrol Boat X X - -         Patrol Boat X X - -

Enter your move: 1,F
```

```
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

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 -
```

Restrictions on the Program

To run the program, you need to type

python3 Assignment4.py "Player1.txt" "Player2.txt" "Player1.in" "Player2.in"

into the terminal. If missing arguments are entered, an error message will be given. If the file names are written incorrectly or the name entered is not found in the directory, an error message is printed.

There is no control in the input ".txt" files. If it is written incorrectly, you may encounter an error or the game may run incorrectly.

If the row or column number is entered incorrectly in the input ".in" files, the error message is printed and the next move is played.

Evaluate Yourself / Guess Grading

Evaluation	Points	Evaluate Yourself / Guess Grading
Readable Codes and Meaningful Naming	5	**4.5**
Using Explanatory Comments	5	**5**
Efficiency (avoiding unnecessary actions)	5	**4.5**
Function Usage	15	**14**
Correctness, File I/O	30	**30**
Exceptions	20	**20**
Report	20	**17**
There are several negative evaluations	…	**0**