HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

BBM 103 Assignment 4 Report

Erdinç Arıcı – 2210356035

02.01.2023

CONTENTS

ANALYSIS

Battleship is a 2 player thinking and strategy based board game. It is played in an area where each player hides his fleet of ships from the other player, where the horizontal and vertical coordinates are predetermined. Players take turns trying to locate and sink the other player's ships by specifying coordinates and shooting. Whichever player sinks the other side's ships sooner wins. There is also a draw situation in the game.

DESIGN

The beginning of everything is to get the necessary entries from the txt file for the codes to work.

I performed this operation in the function called *main_function()*. First it looks for all the required txt files in its location. If it can't find it, it gives an error message for the files it can't find and ends the game. If it finds it, it directs the received inputs to the necessary functions.

I thought a few functions were needed to play the game and I created functions named *location()*, *check_inputs()*, *game()*, *change_board()*, *create_board()*. Apart from these, I created a function called write() that prints the outputs obtained from these functions both to the command line and to the txt file named “Battleship.out”.

The function named *location()* writes the positions of the pre-positioned ships in the given input file into a dictionary. If the given txt file is not in the correct format, it will print the error message and end the game.

The function called *check_inputs()* checks whether the ships in the given txt file are sufficient. If it is not sufficient, it ends the game.

The function called *game()* executes the necessary functions in order to print game boards on the screen.It also checks the game ending states.

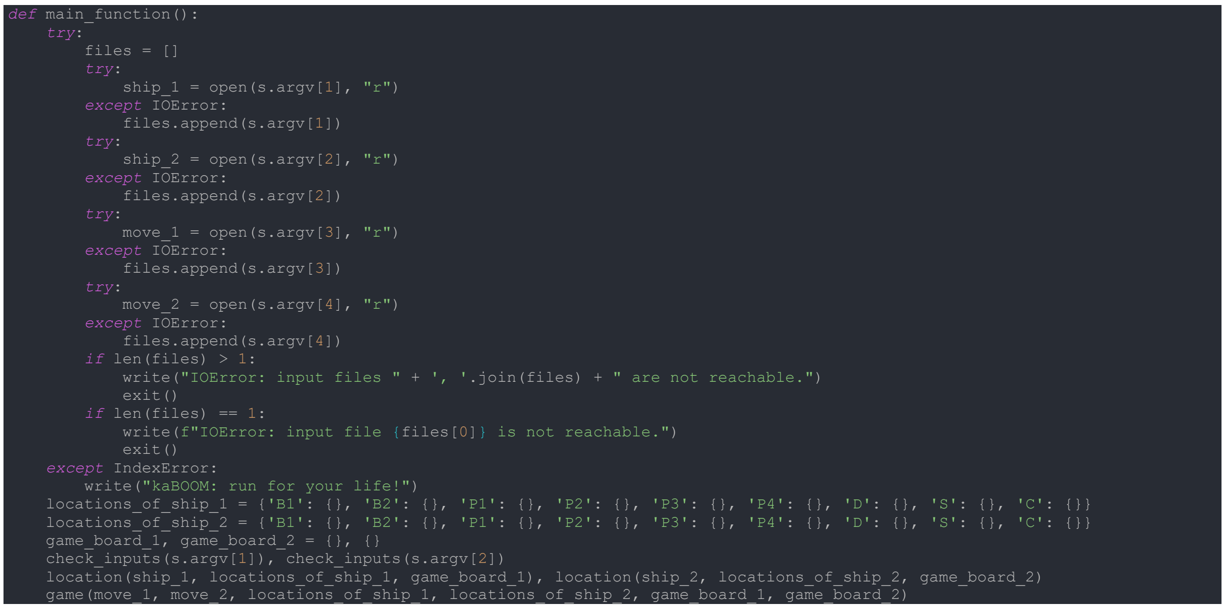
The function called *change_board()* makes the arrangements on the game tables according to the inputs entered. It also checks whether the entered moves are in the correct format. If it's not in the correct form, it skips the incorrect move and runs the code for the next move.

The function called *create_board()* prints the boards of the game.

The *write()* function prints the outputs of other functions to both the command line and the txt file.

PROGRAMMER'S CATALOGUE

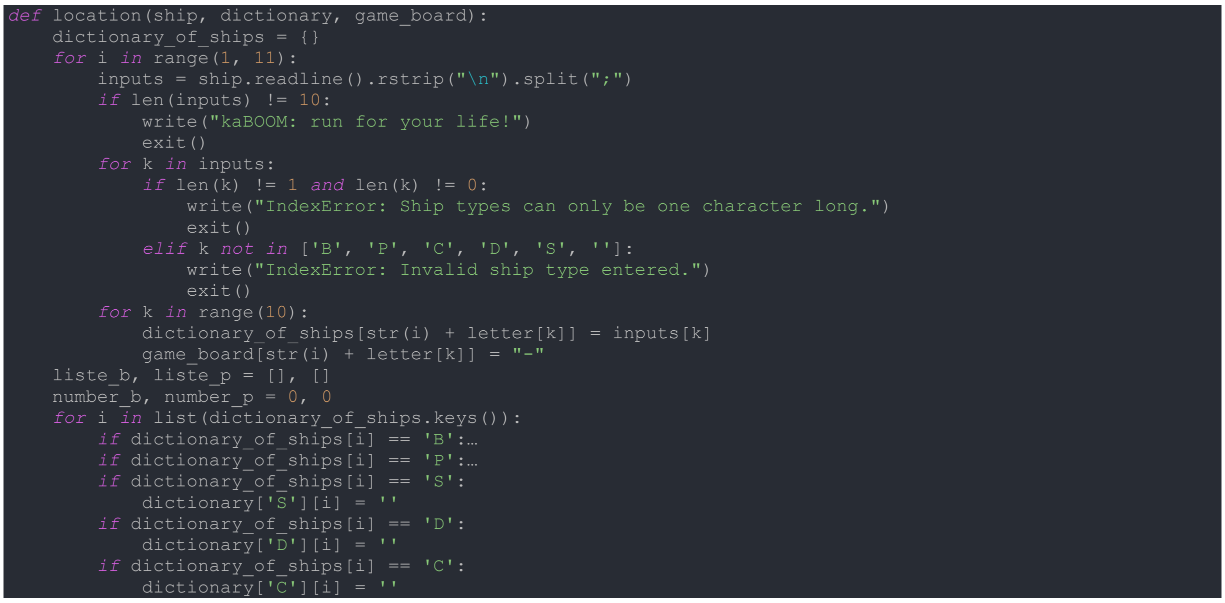
main_function()



```python
def main_function():
    try:
        files = []
        try:
            ship_1 = open(s.argv[1], "r")
        except IOError:
            files.append(s.argv[1])
        try:
            ship_2 = open(s.argv[2], "r")
        except IOError:
            files.append(s.argv[2])
        try:
            move_1 = open(s.argv[3], "r")
        except IOError:
            files.append(s.argv[3])
        try:
            move_2 = open(s.argv[4], "r")
        except IOError:
            files.append(s.argv[4])
        if len(files) > 1:
            write("IOError: input files " + ', '.join(files) + " are not reachable.")
            exit()
        if len(files) == 1:
            write(f"IOError: input file {files[0]} is not reachable.")
            exit()
    except IndexError:
        write("kaBOOM: run for your life!")
    locations_of_ship_1 = {'B1': {}, 'B2': {}, 'P1': {}, 'P2': {}, 'P3': {}, 'P4': {}, 'D': {}, 'S': {}, 'C': {}}
    locations_of_ship_2 = {'B1': {}, 'B2': {}, 'P1': {}, 'P2': {}, 'P3': {}, 'P4': {}, 'D': {}, 'S': {}, 'C': {}}
    game_board_1, game_board_2 = {}, {}
    check_inputs(s.argv[1]), check_inputs(s.argv[2])
    location(ship_1, locations_of_ship_1, game_board_1), location(ship_2, locations_of_ship_2, game_board_2)
    game(move_1, move_2, locations_of_ship_1, locations_of_ship_2, game_board_1, game_board_2)
```

This function first checks the arguments written to the terminal. If there is no txt file for the name of the arguments entered in the terminal, it reports this and stops the code. In addition, if the number of arguments entered into the terminal is less than four, it stops the code. It then sends the inputs from the files to the necessary functions to process them.

location()



```python
def location(ship, dictionary, game_board):
    dictionary_of_ships = {}
    for i in range(1, 11):
        inputs = ship.readline().rstrip("\n").split(";")
        if len(inputs) != 10:
            write("kaBOOM: run for your life!")
            exit()
        for k in inputs:
            if len(k) != 1 and len(k) != 0:
                write("IndexError: Ship types can only be one character long.")
                exit()
            elif k not in ['B', 'P', 'C', 'D', 'S', '']:
                write("IndexError: Invalid ship type entered.")
                exit()
        for k in range(10):
            dictionary_of_ships[str(i) + letter[k]] = inputs[k]
            game_board[str(i) + letter[k]] = "-"
    liste_b, liste_p = [], []
    number_b, number_p = 0, 0
    for i in list(dictionary_of_ships.keys()):
        if dictionary_of_ships[i] == 'B':…
        if dictionary_of_ships[i] == 'P':…
        if dictionary_of_ships[i] == 'S':
            dictionary['S'][i] = ''
        if dictionary_of_ships[i] == 'D':
            dictionary['D'][i] = ''
        if dictionary_of_ships[i] == 'C':
            dictionary['C'][i] = ''
```

This function first checks the txt file where the ship layouts are given. If the given format is incorrect, it stops the code. It creates a dictionary that we will then use to create the table. It assigns all positions to this dictionary and the variable *"X"* to each position.

The main function of this function is to take the positions of the ships separately. For Submarine, Carrier and Destroyer, this job is simple, but for Patrol Boat and Battleship it is much more complex. The algorithm I think for them is to check the right, left, top and bottom of the position by looking at each P and B letter. If there is progress towards only one side of this position, it takes the position in that direction.

check_inputs()

```
def check_inputs(argv):
    list_of_inputs = []
    txt_open = open(argv, "r")
    for i in range(10):
        list_of_inputs.extend(txt_open.readline().rstrip("\n").split(";"))
    if list_of_inputs.count("D") != 3 or list_of_inputs.count("C") != 5 or 
list_of_inputs.count("S") != 3 or list_of_inputs.count("P") != 8 or 
list_of_inputs.count("B") != 8:
        write("IndexError: The ships in the file are not the required number.")
        exit()
```

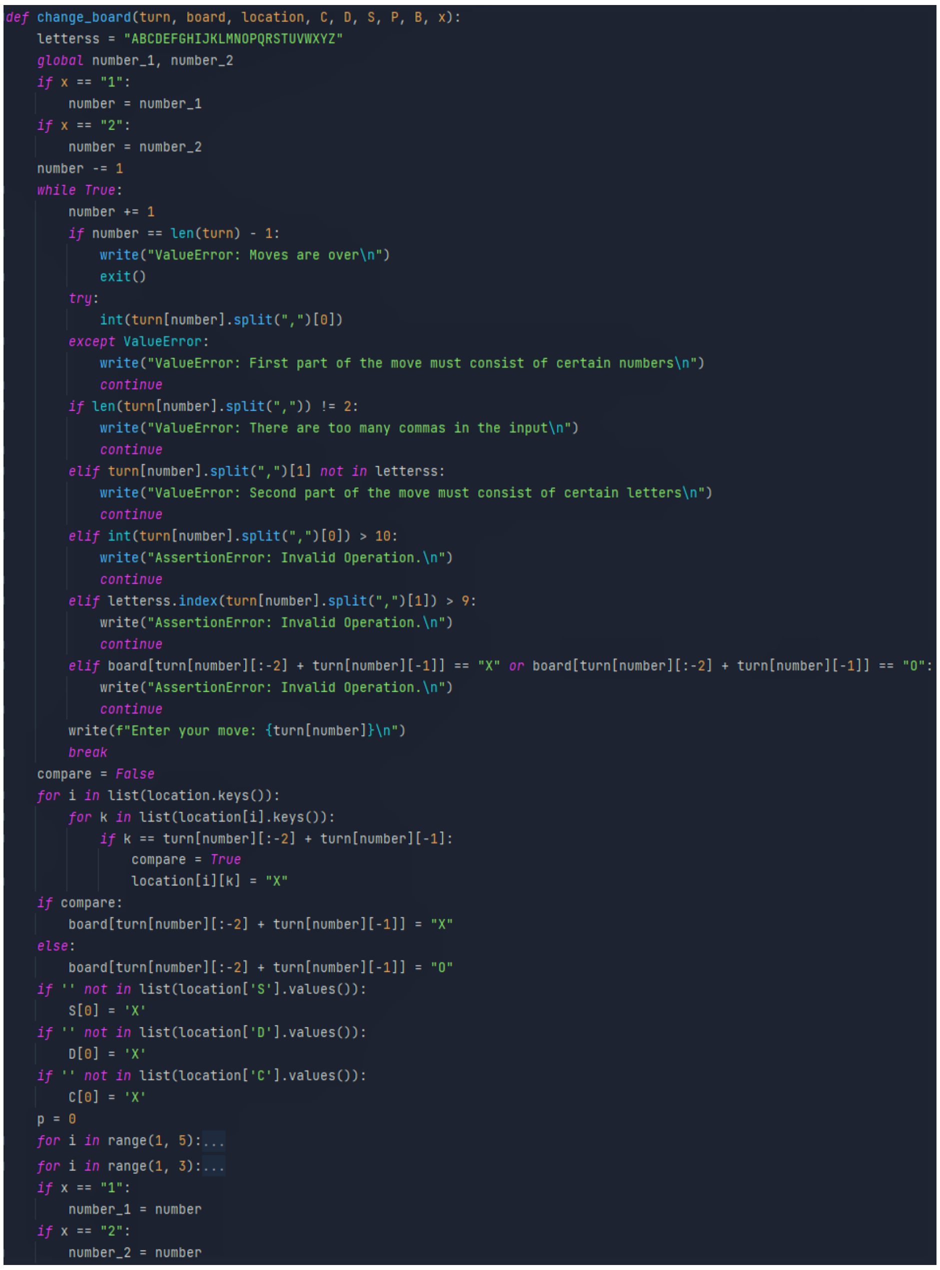
This function checks the number and length of ships in the txt file where the locations of the ships are given. Stops the code if there is an error.

game()

```
number_1, number_2 = 0, 0
def game(move_1, move_2, location_1, location_2, game_board_1, game_board_2):
    global number_1, number_2
    P1, B1, C1, D1, S1 = ["-", "-", "-", "-"], ["-", "-"], ["-"], ["-"], ["-"]
    P2, B2, C2, D2, S2 = ["-", "-", "-", "-"], ["-", "-"], ["-"], ["-"], ["-"]
    player1_turn = move_1.readline().rstrip("\n").split(";")
    player2_turn = move_2.readline().rstrip("\n").split(";")
    number_of_round = 0
    for i in range(2, len(player1_turn) + len(player2_turn)):
        if i % 2 == 0:
            x = "1"
            number_of_round += 1
            write("Player1's Move\n")
            write(f"Round : {number_of_round}\t\t\t\t\tGrid Size: 10x10\n")
            write("Player1's Hidden Board\t\tPlayer2's Hidden Board")
            create_board(game_board_1, game_board_2, C1, D1, S1, P1, B1, C2, D2, S2, P2, B2)
            change_board(player1_turn, game_board_2, location_2, C2, D2, S2, P2, B2, x)
            number_1 += 1
        if i % 2 == 1:
            x = "2"
            write("Player2's Move\n")
            write(f"Round : {number_of_round}\t\t\t\t\tGrid Size: 10x10\n")
            write("Player1's Hidden Board\t\tPlayer2's Hidden Board")
            create_board(game_board_1, game_board_2, C1, D1, S1, P1, B1, C2, D2, S2, P2, B2)
            change_board(player2_turn, game_board_1, location_1, C1, D1, S1, P1, B1, x)
            number_2 += 1
            if "-" not in C1 and "-" not in P1 and "-" not in S1 and "-" not in B1 and "-" not in D1 and
               "-" not in C2 and "-" not in P2 and "-" not in S2 and "-" not in B2 and "-" not in D2:
            elif "-" not in C2 and "-" not in P2 and "-" not in S2 and "-" not in B2 and "-" not in D2:
        if "-" not in C1 and "-" not in P1 and "-" not in S1 and "-" not in B1 and "-" not in D1 and
           "-" not in C2 and "-" not in P2 and "-" not in S2 and "-" not in B2 and "-" not in D2:
        elif "-" not in C1 and "-" not in P1 and "-" not in S1 and "-" not in B1 and "-" not in D1:
```

The main purpose of this function is to run the functions required in order to play the game. First, it takes the moves from the txt that the moves are in. *Player1* starts first, then *Player2* continues and iteratively continues. The number of rounds increases with each move and the *create_board()* and *change_board()* functions are executed after each move. Finally, he checks who won the game or if the game ended in a draw. For this, it checks each word in the dictionaries where the positions of the players' ships are located. If there is no *"-"* or *" "* in the word values, all the ships are sunk.

change_board()



```python
def change_board(turn, board, location, C, D, S, P, B, x):
    letterss = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    global number_1, number_2
    if x == "1":
        number = number_1
    if x == "2":
        number = number_2
    number -= 1
    while True:
        number += 1
        if number == len(turn) - 1:
            write("ValueError: Moves are over\n")
            exit()
        try:
            int(turn[number].split(",")[0])
        except ValueError:
            write("ValueError: First part of the move must consist of certain numbers\n")
            continue
        if len(turn[number].split(",")) != 2:
            write("ValueError: There are too many commas in the input\n")
            continue
        elif turn[number].split(",")[1] not in letterss:
            write("ValueError: Second part of the move must consist of certain letters\n")
            continue
        elif int(turn[number].split(",")[0]) > 10:
            write("AssertionError: Invalid Operation.\n")
            continue
        elif letterss.index(turn[number].split(",")[1]) > 9:
            write("AssertionError: Invalid Operation.\n")
            continue
        elif board[turn[number][:-2] + turn[number][-1]] == "X" or board[turn[number][:-2] + turn[number][-1]] == "O":
            write("AssertionError: Invalid Operation.\n")
            continue
        write(f"Enter your move: {turn[number]}\n")
        break
    compare = False
    for i in list(location.keys()):
        for k in list(location[i].keys()):
            if k == turn[number][:-2] + turn[number][-1]:
                compare = True
                location[i][k] = "X"
    if compare:
        board[turn[number][:-2] + turn[number][-1]] = "X"
    else:
        board[turn[number][:-2] + turn[number][-1]] = "O"
    if '' not in list(location['S'].values()):
        S[0] = 'X'
    if '' not in list(location['D'].values()):
        D[0] = 'X'
    if '' not in list(location['C'].values()):
        C[0] = 'X'
    p = 0
    for i in range(1, 5):...
    for i in range(1, 3):...
    if x == "1":
        number_1 = number
    if x == "2":
        number_2 = number
```

This function first checks if the moves are entered in the correct format. If the entered move is wrong, it skips the wrong move and checks the other move. This loop continues until it finds moves in the correct format.

After finding the move in the correct format, he performs the shot. .Then it writes the letter *"X"* or *"O"* depending on whether the shot is made into the water or the ship.

Finally, this function edits the table showing sunk ships.

create_board()

```
def create_board(game_board_1, game_board_2, C1, D1, S1, P1, B1, C2, D2, S2, P2, B2):
    letters = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
    write("  " + ' '.join(letters) + "\t\t  " + ' '.join(letters))
    for i in range(1, 11):
        write_txt.write(f"{i}")
        print(f"{i}", end="")
        for k in range(10):
            if i == 10 and k == 0:
                write_txt.write(game_board_1[str(i) + letter[k]])
                print(game_board_1[str(i) + letter[k]], end="")
            else:
                write_txt.write(" " + game_board_1[str(i) + letter[k]])
                print(" " + game_board_1[str(i) + letter[k]], end="")
        write_txt.write("\t\t")
        print("\t\t", end="")
        write_txt.write(f"{i}")
        print(f"{i}", end="")
        for k in range(10):
            if i == 10 and k == 0:
                write_txt.write(game_board_2[str(i) + letter[k]])
                print(game_board_2[str(i) + letter[k]], end="")
            else:
                write_txt.write(" " + game_board_2[str(i) + letter[k]])
                print(" " + game_board_2[str(i) + letter[k]], end="")
        write("")
    write("")
    write("Carrier\t\t" + ' '.join(C1) + "\t\t\t\tCarrier\t\t" + ' '.join(C2))
    write("Battleship\t" + ' '.join(B1) + "\t\t\t\tBattleship\t" + ' '.join(B2))
    write("Destroyer\t" + ' '.join(D1) + "\t\t\t\tDestroyer\t" + ' '.join(D2))
    write("Submarine\t" + ' '.join(S1) + "\t\t\t\tSubmarine\t" + ' '.join(S2))
    write("Patrol Boat\t" + ' '.join(P1) + "\t\t\tPatrol Boat\t" + ' '.join(P2) + "\n")
```

The job of this function is to print the board on which the game is played. this function first prints any additional texts to be written outside of the game board. it then enters the dictionary containing that game table, retrieves the information and prints it out. Finally, this function prints the table indicating the ships hit.

write()

```
write_txt = open("Battleship.out", "w")
def write(message):
    print(message)
    write_txt.write(message + "\n")
```

This function prints the outputs from other functions to both the command line and the *Battleship.out* file

REUSABİLİTY

This code will run the Battleship game as long as the ship locations files and move files are in the correct format.

CODE WRITING PROCESS

At first, this assignment scared me because I didn't understand what to do. The next few days were spent thinking about what to do. Then, when I started to write code, it took 7-8 days because I couldn't focus fully on my homework because I had exams. I also encountered a lot of errors, which extended the time. The shortest part for me was writing the report. Because I knew what I had to do and I was thinking about it when I was writing my code.

USER CATALOGUE

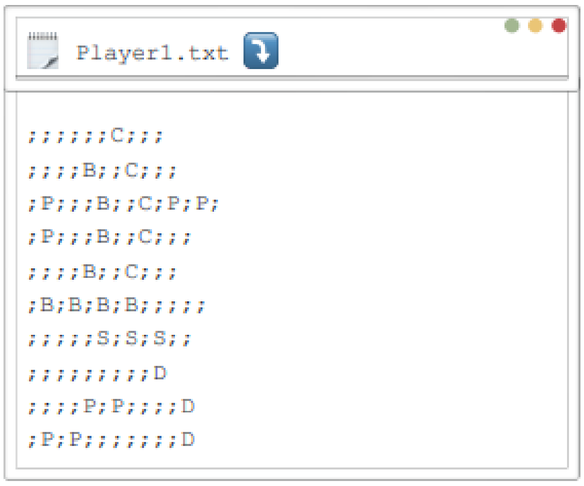
Program’s User Manual

You need to write the place where you want to place the ships and the moves you want to make in the correct form in files with .txt and .in extensions. After doing this, if you type >python3 Assignment4.py "Player1.txt" "Player2.txt" "Player1.in" "Player2.in"< in the terminal, the game will be played.

Restrictions On The Program

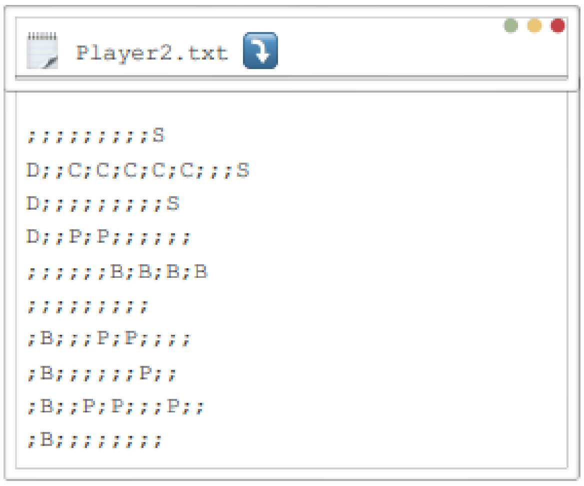
The inputs you write to files with .txt and .in extensions must be in the correct format, otherwise your code will not work.

Sample Input



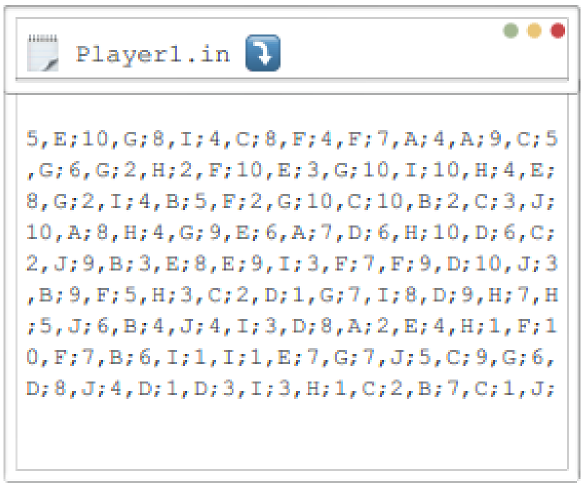
Player1.txt

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



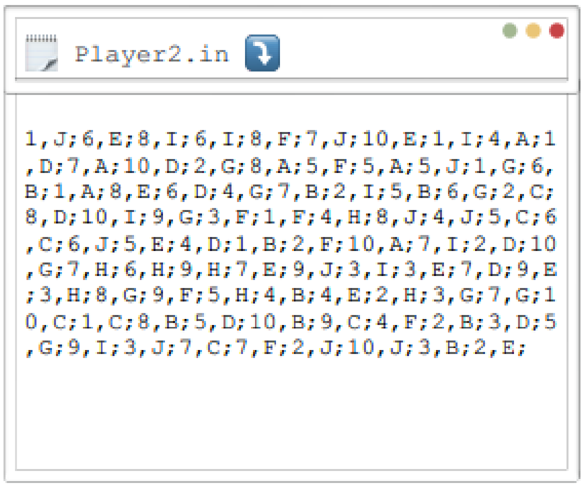
Player2.txt

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



Player1.in

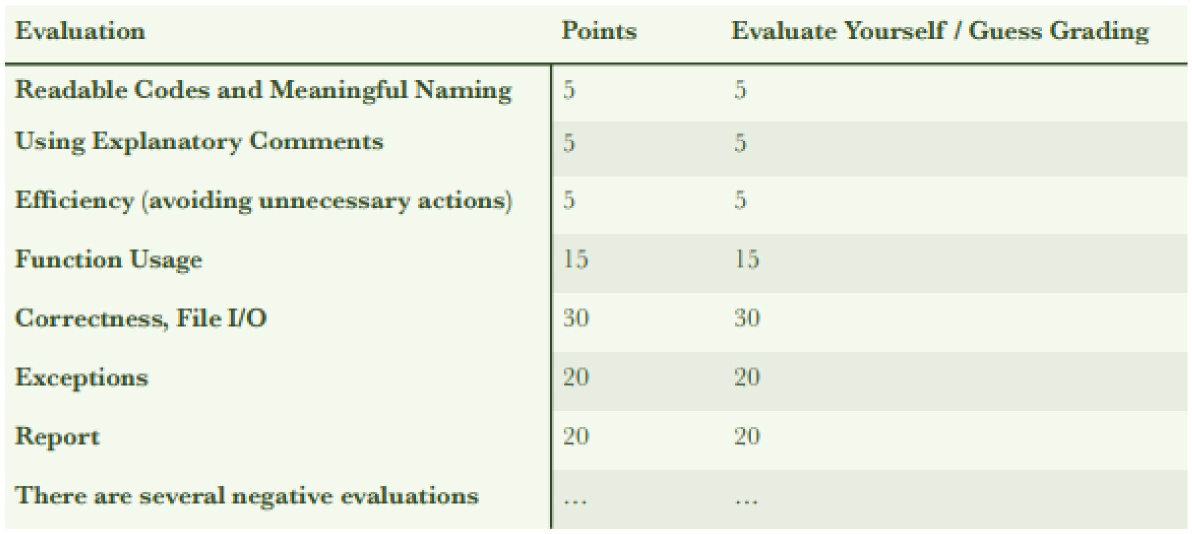
```
5,E;10,G;8,I;4,C;8,F;4,F;7,A;4,A;9,C;5
,G;6,G;2,H;2,F;10,E;3,G;10,I;10,H;4,E;
8,G;2,I;4,B;5,F;2,G;10,C;10,B;2,C;3,J;
10,A;8,H;4,G;9,E;6,A;7,D;6,H;10,D;6,C;
2,J;9,B;3,E;8,E;9,I;3,F;7,F;9,D;10,J;3
,B;9,F;5,H;3,C;2,D;1,G;7,I;8,D;9,H;7,H
;5,J;6,B;4,J;4,I;3,D;8,A;2,E;4,H;1,F;1
0,F;7,B;6,I;1,I;1,E;7,G;7,J;5,C;9,G;6,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



Player2.in

```
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;
```

EVALUATE YOURSELF / GUESS GRADING



Evaluation	Points	Evaluate Yourself / 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	30
Exceptions	20	20
Report	20	20
There are several negative evaluations	…	…