HACETTEPE UNIVERSITY DEPARTMENT OF COMPUTER ENGINEERING

BBM103 – ASSİGNMENT 4 REPORT

Gazi Kağan Soysal

2210356050

02/01/2022

CONTENTS

1. ANALYSIS

We want to play a battleship game. For this, we need to get files from the players showing the locations of the ships and the commands they want to play. Using these files we need to print the current maps and the current ship numbers for each round of the game. When the game ends, we need to print who won the final state of the game.

2.DESIGN

We start the code by reading the text files provided to us. In the file where the players place their ships, we keep this data in a dictionary by accepting spaces between each semicolon and taking the places that write letters as ships.

In the file containing the commands of the players, we separate the semicolons and get the left side of the comma as line information and the right side as column information.

If players place ships of the same type adjacent to each other, our code may become confused. For this, we keep the coordinates of each ship in a dictionary by using the optional files provided to us.

We will start playing the game using the dictionaries we created. We'll make changes to other dictionaries using the script sequentially. Every time we change, we will print a table showing the current state.

In doing so, we will need to display the current ship numbers. For this, we will show the available ships under each table. Then we will print out what the next move is.

The game will be over when one of the players has hit all of the other's ships. If player 2 has a chance to draw the game in the same round player 2 will be given a right. When the game is over, the winner of the game and the final version of the tables will be printed. Unlike other tables, this last table will clearly show the unhitched ships of the winning side.

3. PROGRAMMER'S CATALOGUE

3.1)

```
import sys

player1_txt = sys.argv[1]
player2_txt = sys.argv[2]
player1_in = sys.argv[3]
player2_in = sys.argv[4]

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
error_list = "IOError: input file(s) "

try:
    opening_the_file1 = open(player1_txt, "r")
except IOError:
    error_list = error_list + player1_txt
try:
    opening_the_file2 = open(player2_txt, "r")
except IOError:
    error_list = error_list + player2_txt + ", "
try:
    a = open(player1_in, "r").readlines()
except IOError:
    error_list = error_list + player1_in  + ", "
try:
    b = open(player2_in, "r").readlines()
except IOError:
    error_list = error_list + player2_in + ", "

if len(error_list) > 23:
    error_list = error_list + " is/are not reachable."
    print(error_list)
    error_list = error_list[:-24] + error_list[-22:]
    output_file(error_list)
    exit()
```

Here we read the files given to us and if the filenames are not as we want, we give an error and close the code.

3.2)

```
map_player1 = {}
map_player2 = {}
hidden_player1 = {}
hidden_player2 = {}
ships_p1 = {"Carrier": {}, "Destroyer": {}, "Submarine": {}}
ships_p2 = {"Carrier": {}, "Destroyer": {}, "Submarine": {}}
optional_list1 = []
optional_list2 = []
command_list_p1 = []
command_list_p2 = []
ship_letter_list_C = []
ship_letter_list_D = []
ship_letter_list_S = []
player1loc_for_lenght = open(player1_txt, "r").readlines()
player2loc_for_lenght = open(player2_txt, "r").readlines()
```

Here we create the dictionaries we will use for maps and ship positions.

3.3)

```
for numbers1p in range(len(player1loc_for_lenght)):
    map_player1[str(numbers1p + 1)] = {}
    hidden_player1[str(numbers1p + 1)] = {}
    for letters1p in range(len(player1loc_for_lenght)):
        letter = letters[letters1p]
        map_player1[str(numbers1p + 1)][letter] = ""
        hidden_player1[str(numbers1p + 1)][letter] = "-"

for a in range(len(player1loc_for_lenght)):
    location1 = opening_the_file1.readline().rstrip().split(";")
    ship_letter_list_C = []
    ship_letter_list_D = []
    ship_letter_list_S = []

    for b in range(len(location1)):
        map_player1[str(a + 1)][letters[b]] = location1[b]
        try:
            if location1[b] == "C":
                ship_letter_list_C.append(letters[b])
                ships_p1["Carrier"][str(a + 1)] = ship_letter_list_C
            if location1[b] == "D":
                ship_letter_list_D.append(letters[b])
                ships_p1["Destroyer"][str(a + 1)] = ship_letter_list_D
            if location1[b] == "S":
                ship_letter_list_S.append(letters[b])
                ships_p1["Submarine"][str(a + 1)] = ship_letter_list_S
        except:
            continue
def ships():
    optional_player1 = open("OptionalPlayer1.txt", "r")
    optional_player2 = open("OptionalPlayer2.txt", "r")
    a = optional_player1.readlines()
    b = optional_player2.readlines()

    for ksg in a:
```

```
        optional_list1.append(ksg.rstrip().split(";"))
    for ksg in b:
        optional_list2.append(ksg.rstrip().split(";"))

    for i in optional_list1:
        colon_index = i[0].index(":")
        comma_index = i[0].index(",")
        if i[0][:colon_index][0] == "B":
            range_number = 4
        elif i[0][:colon_index][0] == "P":
            range_number = 2
        abcd = []
        turn = i[0][comma_index + 1:]
        turn2 = i[0][colon_index + 1:comma_index]
        ships_p1[i[0][:colon_index]] = {}
        if i[1] == "right" or i[1] == "left":
            for j in range(range_number):
                abcd.append(turn)
                ships_p1[i[0][:colon_index]][turn2] = abcd
                if i[1] == "right":
                    turn = letters[letters.index(turn) + 1]
                if i[1] == "left":
                    turn = letters[letters.index(turn) - 1]
        if i[1] == "down" or i[1] == "up":
            for j in range(range_number):
                ships_p1[i[0][:colon_index]][turn2] = []
                ships_p1[i[0][:colon_index]][turn2] = turn
                if i[1] == "down":
                    turn2 = str(int(turn2) + 1)
                if i[1] == "up":
                    turn2 = str(int(turn2) - 1)

    for i in optional_list2:
        colon_index = i[0].index(":")
        comma_index = i[0].index(",")
        if i[0][:colon_index][0] == "B":
            range_number = 4
        elif i[0][:colon_index][0] == "P":
            range_number = 2
        abcd = []
        turn = i[0][comma_index + 1:]
        turn2 = i[0][colon_index + 1:comma_index]
        ships_p2[i[0][:colon_index]] = {}
        if i[1] == "right" or i[1] == "left":
            for j in range(range_number):
                abcd.append(turn)
                ships_p2[i[0][:colon_index]][turn2] = abcd
                if i[1] == "right":
                    turn = letters[letters.index(turn) + 1]
                if i[1] == "left":
                    turn = letters[letters.index(turn) - 1]
        if i[1] == "down" or i[1] == "up":
            for j in range(range_number):
                ships_p2[i[0][:colon_index]][turn2] = []
                ships_p2[i[0][:colon_index]][turn2] = turn
                if i[1] == "down":
                    turn2 = str(int(turn2) + 1)
                if i[1] == "up":
                    turn2 = str(int(turn2) - 1)
ships()
```

With these loops, we map the coordinates of the ships to the ships. Thus, there is no confusion in adjacent placement situations.

3.4)

```
ships_control1 = []
ships_control2 = []
for z in ships_p1:
    ships_control1.append(z[0])
for z in ships_p2:
    ships_control2.append(z[0])
```

In these lists, there is information about which ship and how many. When each ship is destroyed, it is deleted from this list, and using this list we can show how many ships are left.

3.5)

```
def commands():
    c = a[0].split(";")
    f = b[0].split(";")

    for d in c:
        command_list_p1.append(d.split(","))
    command_list_p1.pop()
    for e in f:
        command_list_p2.append((e.split(",")))
    command_list_p2.pop()
commands()
```

With this function, we keep a list of the commands of the players.

3.6)

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

def drawing_table(command):
    output_file(f"Player{int(command) % 2 + 1}'s Move\n")
    print(f"Round: {int(command/2 + 1)}", end = "\t\t\t")
    output.write(f"Round : {int(command/2 + 1)}" + "\t\t\t\t\t")
    output_file(f"Grid Size:
{len(player1loc_for_lenght)}X{len(player1loc_for_lenght)}\n")
    output_file("Player1's Hidden Board\t\tPlayer2's Hidden Board")
    print("  ",end="")
    output.write("  ")
    for number_p1 in hidden_player1:
        for letter_p1 in hidden_player1[number_p1]:
            print(letter_p1, end=" ")
            output.write(letter_p1 + " ")
        print("\t\t", end="  ")
        output.write("\t\t" + "  ")
        break

    for number_p2 in hidden_player2:
        for letter_p2 in hidden_player2[number_p2]:
            print(letter_p2, end=" ")
            output.write(letter_p2 + " ")
        output_file("")
        break

    def p1board(number_to_write_p1):
        if number_to_write_p1 < 10:
            print(number_to_write_p1, end=" ")
            output.write(str(number_to_write_p1) + " ")
        else:
            print(number_to_write_p1, end="")
            output.write(str(number_to_write_p1))
        for marks_p1 in hidden_player1[str(number_to_write_p1)]:
            print(hidden_player1[str(number_to_write_p1)][marks_p1], end="
")
            output.write(hidden_player1[str(number_to_write_p1)][marks_p1]
+ " ")
        p2board(number_to_write_p1)

    def p2board(number_to_write_p2):
        print("\t\t", end="")
        output.write("\t\t")
        if number_to_write_p2 < 10:
            print(number_to_write_p2, end=" ")
            output.write(str(number_to_write_p2) + " ")
        else:
            print(number_to_write_p2, end="")
            output.write(str(number_to_write_p2))
        for letter_to_write_p2 in hidden_player2[str(number_to_write_p2)]:

print(hidden_player2[str(number_to_write_p2)][letter_to_write_p2], end=" ")

output.write(hidden_player2[str(number_to_write_p2)][letter_to_write_p2] +
" ")
        print()
        output.write("\n")
        if number_to_write_p2 < len(location1):
```

```
            p1board(number_to_write_p2 + 1)

    number_to_write_p1 = 1
    p1board(number_to_write_p1)
    print()
    output.write("\n")
```

We call this function after each move is made. In this function, we create tables by showing the places that were hit.

3.7)

```
def execute1():
    global number_target1, letter_target1
    if map_player2[number_target1][letter_target1] == "":
        hidden_player2[number_target1][letter_target1] = "O"
    else:
        hidden_player2[number_target1][letter_target1] = "X"

    for y in list(ships_p2):
        for v in list(ships_p2[y]):
            if type(ships_p2[y][v]) == list:
                if v == number_target1 and letter_target1 in
ships_p2[y][v]:
                    if len(ships_p2[y][v]) > 1:
                        ships_p2[y][v].remove(letter_target1)
                    elif len(ships_p2[y][v]) == 1:
                        ships_p2[y].pop(v)
            else:
                if v == number_target1 and ships_p2[y][v] ==
letter_target1:
                    ships_p2[y].pop(v)
```

In this function, we apply the information we get from the command list to other dictionaries and make the code progress dynamically. We have another function where we apply the same operation in the 2nd player.

3.8)

```
def count_of_ships():
    for ships in ["Carrier", "Battleship", "Destroyer", "Submarine",
"Patrol Boat"]:
        if ships == "Carrier":
            print(ships, end=" \t")
            output.write(ships + " \t")
        else:
            print(ships, end="\t")
            output.write(ships + "\t")

        t = ships_control1.count(ships[0])
        f = ships_control2.count(ships[0])
        if ships == "Carrier" or ships == "Destroyer" or ships ==
"Submarine":
            print((1-t)*"X " + t*"- ", end="\t\t")
            output.write((1-t)*"X " + t*"- " + "\t\t\t\t")
        elif ships == "Battleship":
            print((2-t)*"X " + t * "- ", end="\t\t")
            output.write((2 - t) * "X " + t * "- " + "\t\t\t")
        else:
            print((4-t) * "X " + t * "- ", end="\t")
            output.write((4 - t) * "X " + t * "- " + "\t\t")

        if ships == "Carrier":
            print(ships, end="\t\t")
            output.write(ships + "\t\t")
        else:
            print(ships, end="\t")
            output.write(ships + "\t")

        if ships == "Carrier" or ships == "Destroyer" or ships ==
"Submarine":
            output_file((1-f)*"X " + f*"- ")
        elif ships == "Battleship":
            output_file((2-f)*"X " + f * "- ")
        else:
            output_file((4-f) * "X " + f * "- ")
```

In this function, we print how many hits from which ship of each player and how many are left.

3.9)

```
def last_write(command):
    for inside1 in list(ships_p1):
        if len(ships_p1[inside1]) == 0:
            if inside1[0] in ships_control1:
                ships_control1.remove(inside1[0])
                ships_p1.pop(inside1)
        else:
            for inside2 in ships_p1[inside1]:
                if type(ships_p1[inside1][inside2]) == list:
                    if len(ships_p1[inside1][inside2]) == 0:
                        if inside1[0] in ships_control1:
                            ships_control1.remove(inside1[0])
                            ships_p1.pop(inside1)

    for inside1 in list(ships_p2):
        if len(ships_p2[inside1]) == 0:
            if inside1[0] in ships_control2:
                ships_control2.remove(inside1[0])
                ships_p2.pop(inside1)
        else:
            for inside2 in ships_p2[inside1]:
                if type(ships_p2[inside1][inside2]) == list:
                    if len(ships_p2[inside1][inside2]) == 0:
                        if inside1[0] in ships_control2:
                            ships_control2.remove(inside1[0])
                            ships_p2.pop(inside1)

    if command % 2 == 0:
        output_file(f"\nEnter your move:
{number_target1},{letter_target1}\n")
    else:
        output_file(f"\nEnter your move:
{number_target2},{letter_target2}\n")
```

In this function, we can find the numbers correctly by removing the hit ships from the list we checked. Then we print whichever command will be played.

3.10)

```
def final_statement():

    if number_x_p1 == 27 and number_x_p2 == 27:
        output_file("It is a Draw!\n")
    elif number_x_p1 == 27:
        output_file("Player2 Wins!\n")
    elif number_x_p2 == 27:
        output_file("Player1 Wins!\n")
    output_file("Final Information\n")

    output_file("Player1's Board\t\t\tPlayer2's Board")
    print("  ", end="")
    output.write("  ")
    for number_p1 in hidden_player1:
        for letter_p1 in hidden_player1[number_p1]:
            print(letter_p1, end=" ")
            output.write(letter_p1 + " ")
        print("\t\t", end="  ")
        output.write("\t\t" + "  ")
        break

    for number_p2 in hidden_player2:
        for letter_p2 in hidden_player2[number_p2]:
            print(letter_p2, end=" ")
            output.write(letter_p2 + " ")
        output_file("")
        break

    if number_x_p1 == 27:
        for inside1 in map_player2:
            for inside2 in map_player2[inside1]:
                if map_player2[inside1][inside2] != "" and
hidden_player2[inside1][inside2] == "-":
                    hidden_player2[inside1][inside2] =
map_player2[inside1][inside2]

    elif number_x_p2 == 27:
        for inside1 in map_player1:
            for inside2 in map_player1[inside1]:
                if map_player1[inside1][inside2] != "" and
hidden_player1[inside1][inside2] == "-":
                    hidden_player1[inside1][inside2] =
map_player1[inside1][inside2]
    count = 1

    def finalp1board(count):
        if count < 10:
            print(count, end=" ")
            output.write(str(count) + " ")
        else:
            print(count, end="")
            output.write(str(count))
        for marks_p1 in hidden_player1[str(count)]:
            print(hidden_player1[str(count)][marks_p1], end=" ")
            output.write(hidden_player1[str(count)][marks_p1] + " ")
        finalp2board(count)
```

```
def finalp2board(number_to_write_p2):
    print("\t\t", end="")
    output.write("\t\t")
    if number_to_write_p2 < 10:
        print(number_to_write_p2, end=" ")
        output.write((str(number_to_write_p2) + " "))
    else:
        print(number_to_write_p2, end="")
        output.write(str(number_to_write_p2))
    for letter_to_write_p2 in hidden_player2[str(number_to_write_p2)]:
        print(hidden_player2[str(number_to_write_p2)][letter_to_write_p2],
end=" ")

output.write(hidden_player2[str(number_to_write_p2)][letter_to_write_p2] +
" ")
    print()
    output.write("\n")
    if number_to_write_p2 < len(location1):
        finalp1board(number_to_write_p2 + 1)
    else:
        print()
        output.write("\n")
finalp1board(count)
count_of_ships()
```

When we come to this function, the game is now finished and we print the result tables and the winning status of the game.

3.11)

```
for k in range(len(max(command_list_p1, command_list_p2))*2):
    try:
        drawing_table(k)

        if k % 2 == 0:
            try:
                number_target1 = command_list_p1[int(int(k) / 2)][0]
                letter_target1 = command_list_p1[int(int(k) / 2)][1]

                assert int(number_target1) <= len(player1loc_for_lenght)
and letter_target1 in map_player1["1"]
                execute1()

                if number_target1 == "" or letter_target1 == "":
                    raise IndexError
                if number_target1 not in map_player1 or letter_target1 not
in map_player1["1"]:
                    raise ValueError

            except AssertionError:
                output_file("AssertionError: Invalid Operation.")
                number_target1 = command_list_p1[int(int(k+2) / 2)][0]
                letter_target1 = command_list_p1[int(int(k+2) / 2)][1]
                execute1()
                command_list_p1.pop(int(k/2))
                last_write(k)
                continue
            except IndexError:
                output_file("IndexError: You must enter your commands in
number comma letter semicolon order!")
                number_target1 = command_list_p1[int(int(k + 2) / 2)][0]
                letter_target1 = command_list_p1[int(int(k + 2) / 2)][1]
                execute1()
                command_list_p1.pop(int(k / 2))
                last_write(k)
                continue
            except ValueError:
                output_file("ValueError: You must enter your commands in
number comma letter semicolon order!")
                number_target1 = command_list_p1[int(int(k + 2) / 2)][0]
                letter_target1 = command_list_p1[int(int(k + 2) / 2)][1]
                execute1()
                command_list_p1.pop(int(k / 2))
                last_write(k)
                continue
```

This loop allows us to play the game continuously. When an incorrect command is entered, it gives an error message and continues to read the file until the person making the mistake makes the right move.

Time spent analyzing, designing, implementing, testing, and reporting

On the evening of the assignment, I read the pdf file and thought about what to do with the homework. Then I started writing my code slowly. There were days when I didn't even look. There were also days when I looked for 1 day. On average, I finished my homework by looking at an average of 3 hours a day for 2 weeks.

Reusability

I think my code can be used comfortably by other programmers. Variable names are understandable. In addition, since it is divided into functions, the code can be checked piece by piece.

4. USER CATALOGUE

4.1)Program's user manual/tutorial

To redeem the code, you can first place the ships in the ".txt" files using semicolons as you wish. Then you can write your commands in the ".in" files and run the code. Finally, you can see the entire output of the game by entering the "Battleship.out" file.

4.2)Restrictions on the program

When an incorrect move is entered, an extra 1 correct move must be entered instead. Otherwise, the number of moves will not be equal and the side that does not make a mistake cannot play its last move.

GRADING TABLE

Evaluation	Points	Evaluate Yourself / Guess Grading
Readable Codes and Meaningful Naming	5	5

Evaluation	Points	Evaluate Yourself / Guess Grading
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	…	0