|  |
| --- |
| **Assignment 4: Battle of Ships** |
| ***Name****:* Yusuf İpek ***SN****:* 222035048 |
| *BBM103* |
| **Delivery Date:** 03.01.2023 |

Contents

[Analysis 3](#_Toc123679705)

[Design 3](#_Toc123679706)

[Classes: 3](#_Toc123679707)

[Lists: 3](#_Toc123679708)

[Dictionaries: 3](#_Toc123679709)

[Functions: 3](#_Toc123679710)

[MainProgram 3](#_Toc123679711)

[Get Arguments 4](#_Toc123679712)

[Check Input 4](#_Toc123679713)

[Read Input File 4](#_Toc123679714)

[Create Output File 4](#_Toc123679715)

[Write the Outputs 4](#_Toc123679716)

[Split Start Positions 4](#_Toc123679717)

[Split Moves 4](#_Toc123679718)

[Check Moves 4](#_Toc123679719)

[Game Round 4](#_Toc123679720)

[Print Boards 4](#_Toc123679721)

[Hit Boat 4](#_Toc123679722)

[Game Over 4](#_Toc123679723)

[Programmer’s Catalogue 5](#_Toc123679724)

[Imports 5](#_Toc123679725)

[Defining Constants 5](#_Toc123679726)

[Main Program 5](#_Toc123679727)

[Get Arguments Funciton 6](#_Toc123679728)

[File Functions 6](#_Toc123679729)

[readInputFile Function 6](#_Toc123679730)

[createOutputFile Function 6](#_Toc123679731)

[writeFile Function 6](#_Toc123679732)

[Spliting Inputs Functions 7](#_Toc123679733)

[splitStartPositions Function 7](#_Toc123679734)

[splitMoves Function 7](#_Toc123679735)

[checkInputs Function 7](#_Toc123679736)

[Player Class 8](#_Toc123679737)

[Initialization and createPlayer Funciton 8](#_Toc123679738)

[getBoatsIndex Function 8](#_Toc123679739)

[checkBoats Function 8](#_Toc123679740)

[groupingBoats Function 9](#_Toc123679741)

[checkRowOrColumn Function 9](#_Toc123679742)

[printBoards Function 10](#_Toc123679743)

[printBoard Function 10](#_Toc123679744)

[printBoats Function 10](#_Toc123679745)

[gameRound Function 11](#_Toc123679746)

[hitBoat Function 12](#_Toc123679747)

[gameOver Function 12](#_Toc123679748)

[checkMove Function 12](#_Toc123679749)

[User Catalogue 13](#_Toc123679750)

[File Hierarchy Examples 13](#_Toc123679751)

[Txt Files Examples (Positions) 13](#_Toc123679752)

[In Files Example (Moves) 14](#_Toc123679753)

[Outputs Example 14](#_Toc123679754)

[Grading Table 15](#_Toc123679755)

# 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 begining. 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.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

*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 repats itself write the proper error mesage 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

Import the modules.

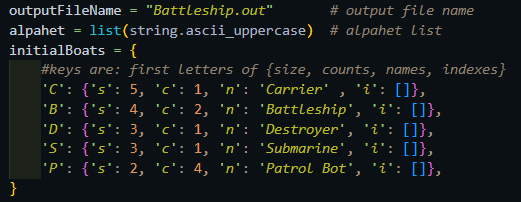
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Import Modules Codes*

* 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 alphatet list
* COPY for cloning a dictionary without coping its memory address just copy the values.

## Defining Constants

*Constant Variables Codes*

Define the constant variables,

* Alpahet List
* Output file name as “Battleship.out”
* Boats’ informations: Names, counts, size and indexes

## Main Program

*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 Funciton

Return Type*:* *boolean*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu *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

Parameter: string *fileName*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Read Input File Function Codes*

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

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu*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

Paramaters: string *text,* string *end*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Write Output Function Codes*

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

## Spliting Inputs Functions

### splitStartPositions Function

Parameter: List *dataList*

Return Type: List

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Split Start Positions Function Codes*

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

Split the list by “;” charachter and change the empty charchters with “-” charachter 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

Get the input data with readInputFile function as a list.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu*Split Moves Function Codes*

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

## checkInputs Function

Return Type: boolean

* Create a list which has the input files names.

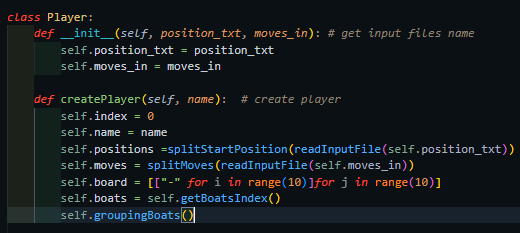
metin, ekran görüntüsü, ekran içeren bir resim

Açıklama otomatik olarak oluşturuldu *Check Inputs Function Codes*

* Define a *string* variable for wrong inputs.
* Call *readInput* function for each name in the inputs. If there is an error the *readInput* funtion 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

### Initialization and createPlayer Funciton



*Player Class Initialization and Create Player Function Codes*

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

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*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

In the *Player* Class,

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Check Boats Function Codes*

* 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

In the *Player* class,

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Grouping Boats Function Codes*

* 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

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

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Check Row Or Column Function Codes*

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

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*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



*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.)

### printBoats Function

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Print Boats Function Codes*

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.

* Create a list which has the players (player1 and player2).

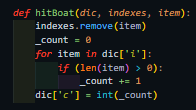
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Game Round Function Codes*

* 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 whit 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



*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 positons of components of boats’.
* Update the count of it.

## gameOver Function

Parameters: string *playerName*, boolean *isDraw*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu*Game Over Function Codes*

* 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

Parameter: Player *player*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Check Move Function Codes*

* Get the move by using player’s moves list and player’s index.
* Split the move by “,” looking char.
* Check for the lenght if is less than expected raise index error, if it is grater 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")*

*Execution Command Example*

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

## File Hierarchy Examples

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

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu*File Hierarchy After Execution*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*File Hierarchy Before Execution*

## Txt Files Examples (Positions)

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu*Players Ships Positions Example (Player2.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.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Players Ships Positions Example (Player1.txt)*

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 !



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

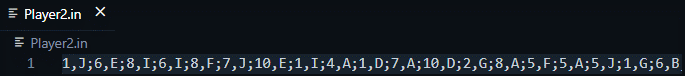
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

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

## In Files Example (Moves)

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

*Input File Example (player2’s 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.

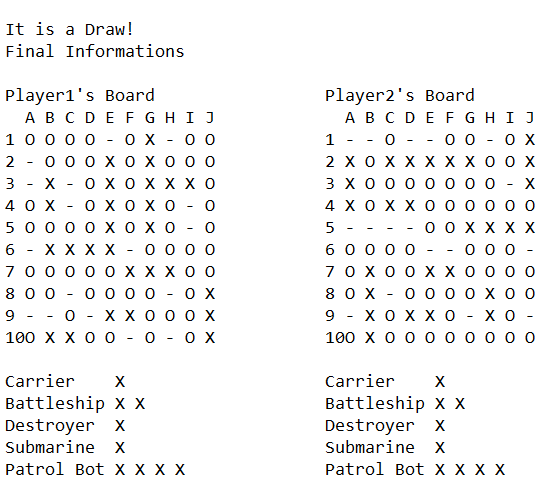
If the form is not index,letter program write value error, if the move has just one letter or integer progrom write index error. The move have more than one move like ;2,A,B; this cause a value error (when the move components is more than 2). Also you should write less than 11 for integer and write a 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.

metin içeren bir resim

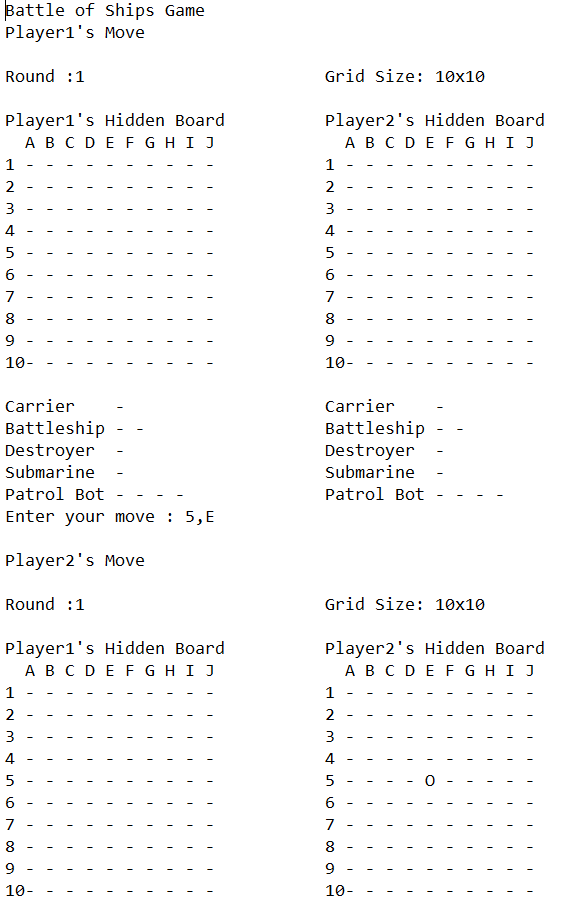
Açıklama otomatik olarak oluşturuldu

*Wrong Input Example*

## Outputs Example



*Output File Example (For Draw)*



*Output File Example*

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

*Console Output Example*

# 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 |