ASSIGNMENT 4

BBM 101

Deniz Kutay Açıcı – 2220356022

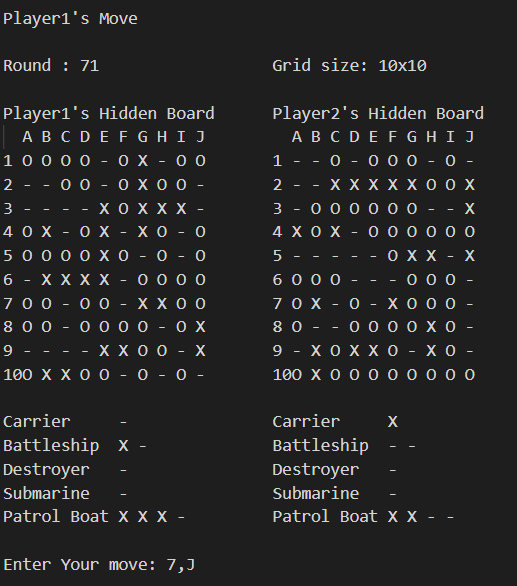
03.01.2023



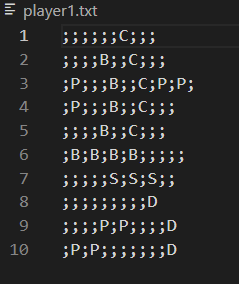
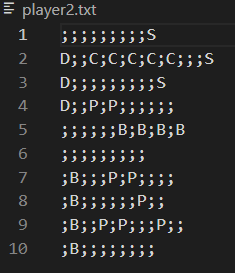
**Analysis**

In Assignment, we have to design a battleship game.Battleship is a well-known strategy and guessing game for two players. At the beginning of the game two players places their ships into a 10x10 game area.The ships are carrier(size:5,count:1),battleship(size:4,count:2),destroyer(size:3,count:1),submarine(size:3,count:1) and patrol boat(size:2,count:4).After placements of ships players start to bomb other player’s grid in turn.If bomb hits a ship, the square becomes “X”, otherwise it becomes “O”. When a player sunks all the other player’s ships the game over at that round. If both side’s all ships sunk, game is tie.

**Design**

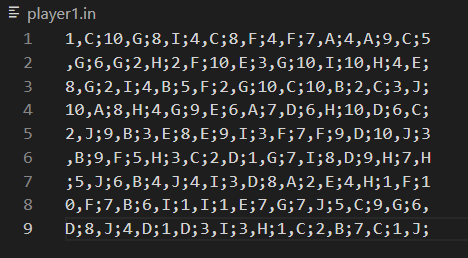


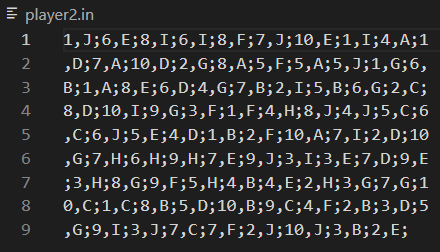
The output must be like the picture which is at above.We need to create a grid for both players with size 10 x 10. Then we replace the ships according to users input which is shown at the bottom.

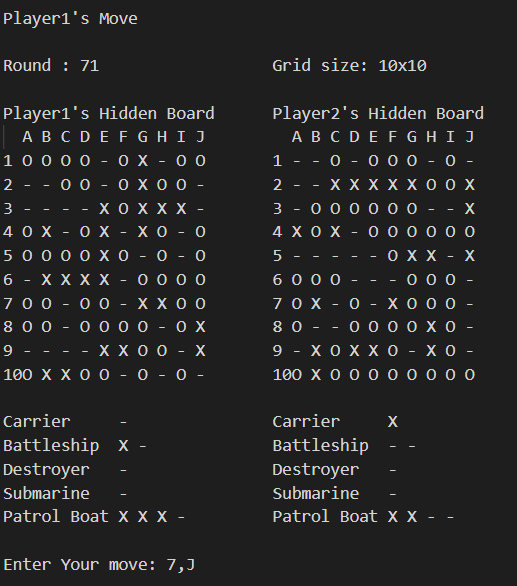
Semicolons represents seperation between columns and in every line represents rows.So we need to import that inputs to the program.

After replacements we need the bombing part.For bombing part inputs must be like the pictures at the bottom.





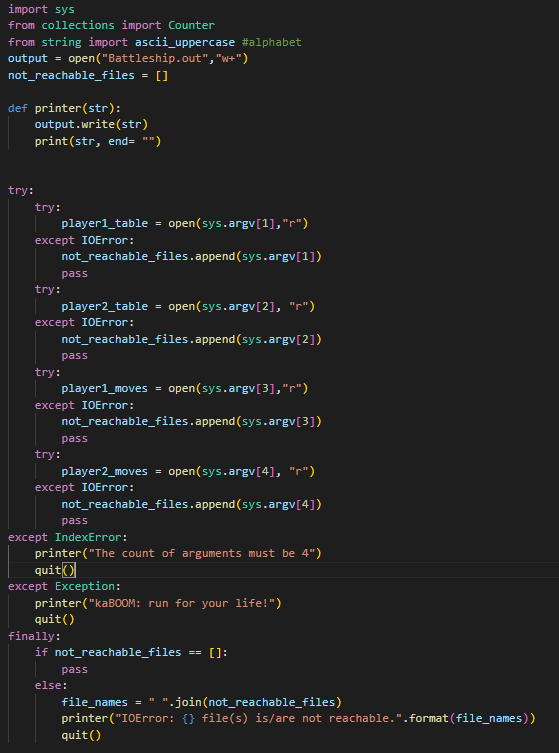
Semicolons represents seperation between rounds, numbers are rows name and letters are columns name. So player1’s first bomb is dropped at 1C square and the second 10G ect. With that bombings grid becomes from “-“ to “O” or “X”.



In bottom of that picture,it shows which ships are sunk. For doing that part we need to group squares according to which ships they are belong to.

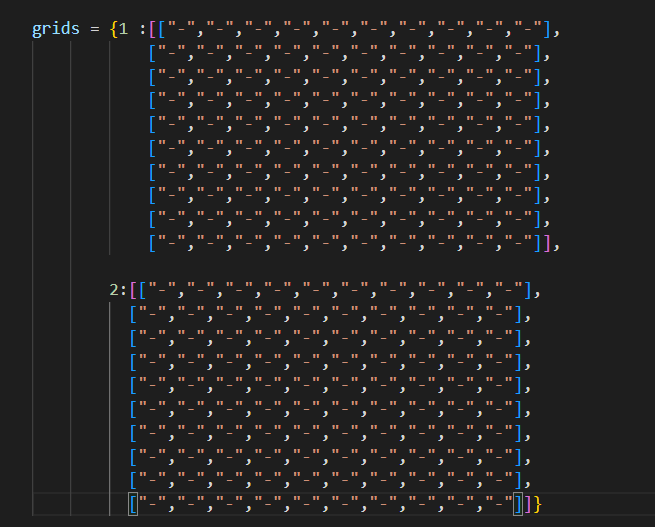
And finally, we have to finish the game when one of the player’s all ships are sunk.

**Coding**

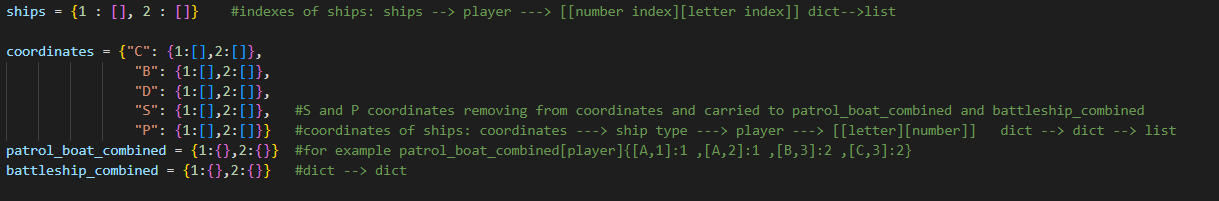


At first the progrom opens “battleship.out” file.If file doesn’t exist it creates one. Printer function is basically print and write to battleship.out the string which you want to print at the same time.

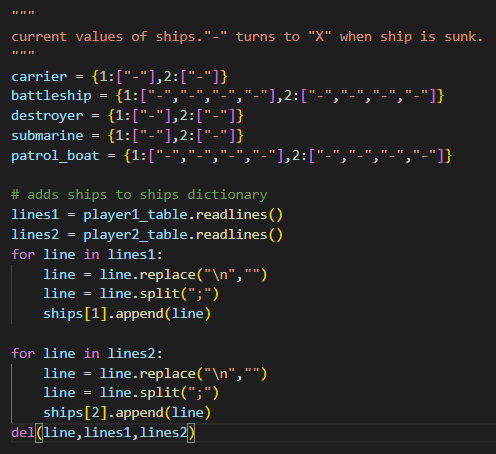
Later that try except chains opens the input files which is needed.(player1.txt, player2.txt, player1.in, player2.in).If an argument doesn’t reachable,argument is adding to the not\_reachable\_files list.Also there is warning messages for IndexError and other errors. Finally It checks the not\_reachable\_files list for the list is empty or not.If list is not empty there is an argument that cannot reachable.So it prints error message and stop running the code.



There is a dictionary called grids. It is basically the grids that shows in the game.”-“ turns to “O” or “X” when the players gets a bomb to that square.

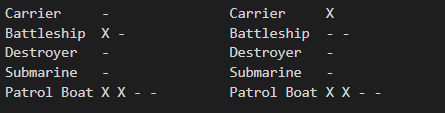


Ships dictionary contains every ships positions as indexes.Later on the coordinates will be appended to coordinates as letter,number format and seperated by ship types. Because of patrol boats and battleships has more than 1 ship they will also be appended to patrol\_boat\_combined and battleship\_combined and will be removed from the coordinates.



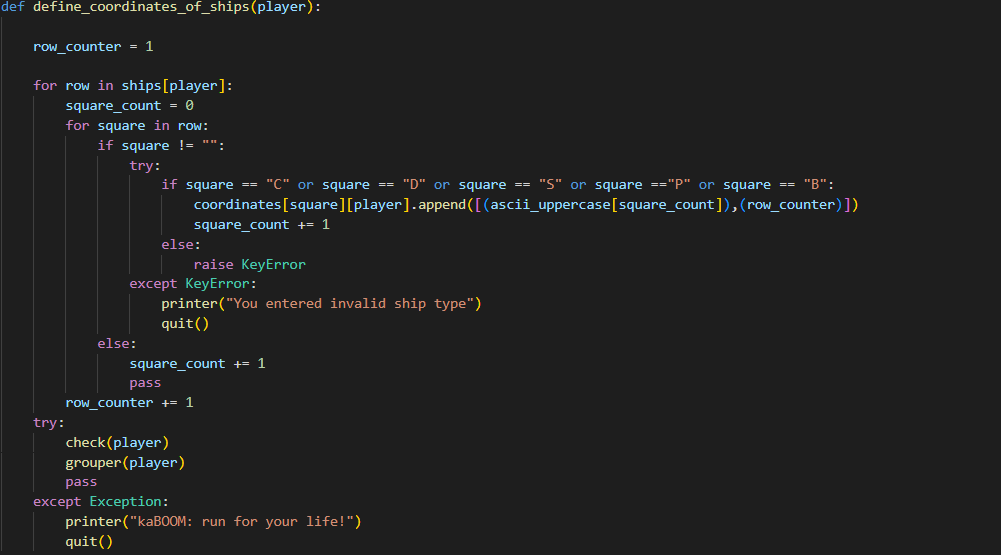
The 5 dictionaries at the top is the current values of ships which will be printed in the game screen .“-“ turns “X” when a ship sunks.

For example:



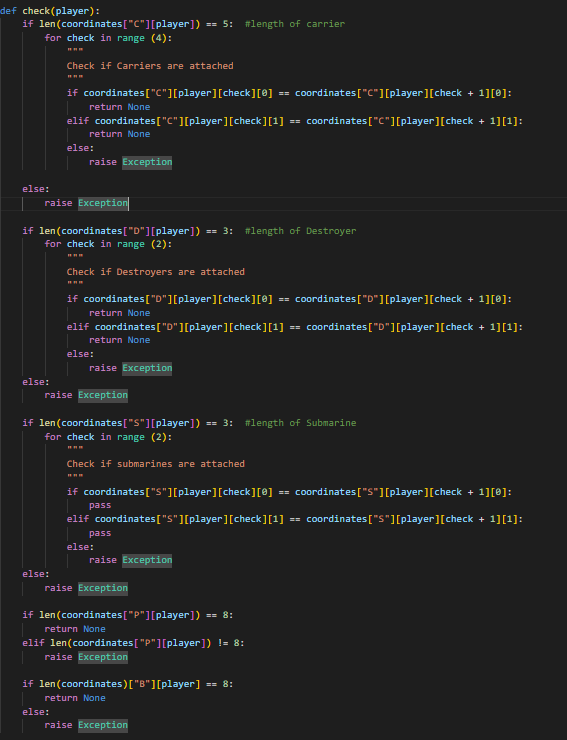
At the bottom part it reads player1.in and player2.in files. It splits the semicolons and append to the ships dictionary. So ships 🡪 player list would be like [['', '', '', '', '', '', 'C', '', '', ''],…]. With the help of these data we can get the ships coordinates with looking indexes of letters.

**Functions**



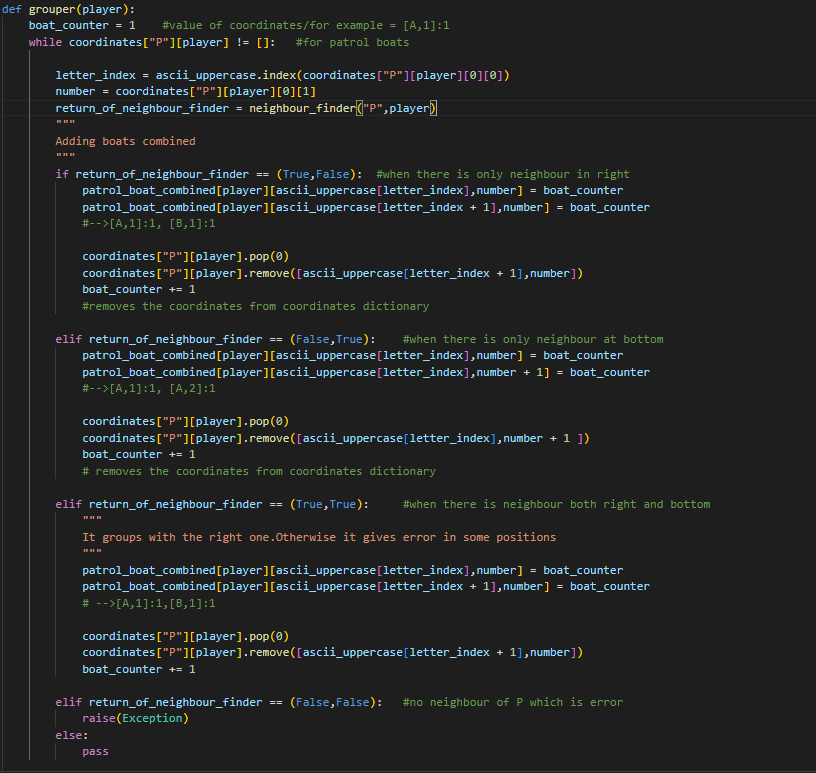
Define\_coordinates\_of\_ships:

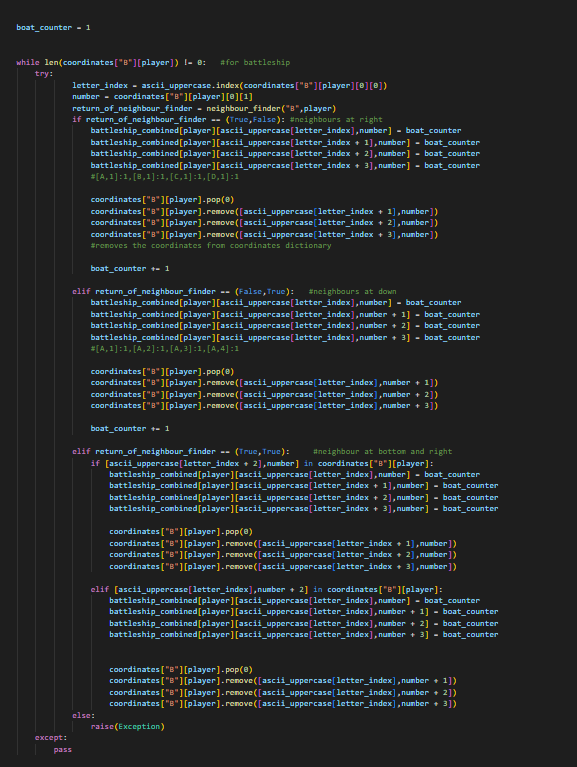
In that function it checks every indexes of ships 🡪 player list. And then the function appends the coordinates to the coordinates file with [letter,number] format. If there is a ship type which is not defined it raises KeyError and quit from the program. Check and grouper functions will be mentioned later.



Check:

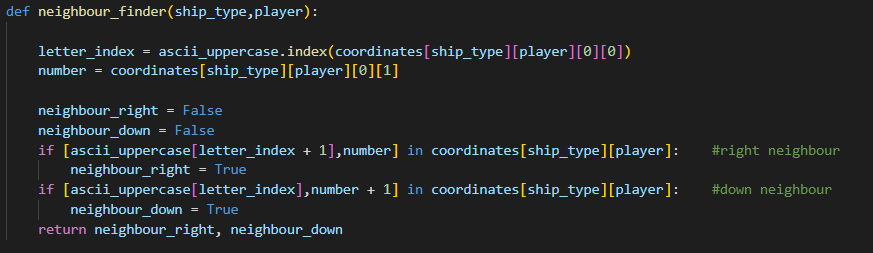
It checks the how many a ship take up squares and they are attached or not. If there is a mistake it raises exception error and the game stops before starting. For carrier,destroyer and submarine it checks the right and down column if that square has that ship it passes and look for another square. It does not do that for Patrol boat and battleship because it will be handled in grouper function.



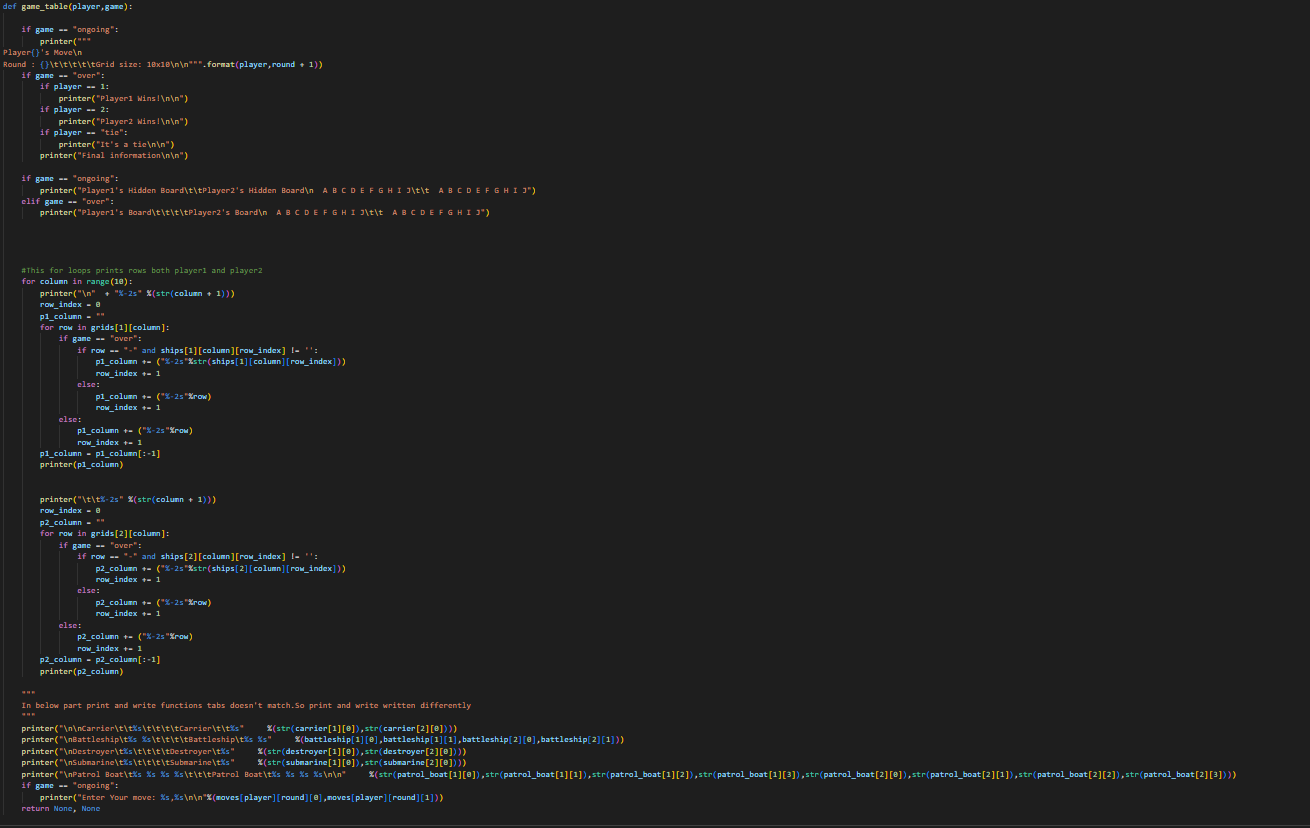


Grouper:

The function takes the coordinates of patrol boats and battleships from coordinates dictionary.Then it checks their right and down neighbours with the help of neighbour\_finder function.For Patrol boats , when there is only one neighbour it appends the coordinates to the patrol\_boat\_combined dictionary with the same value.(value is boat\_counter).When there is neighbours both right and down it appends the first coordinate and the right one’s coordinate to the patrol\_boat\_combined with the same value.It appends with the right one because otherwise there may be unexpected errors.When there is no neighbour it raises error.For battleship it does the same things when there is only one neighbour.But when there is 2 neighbours it looks for 2 right square and 2 down square. Then it combines the battleships same as the patrol boats.

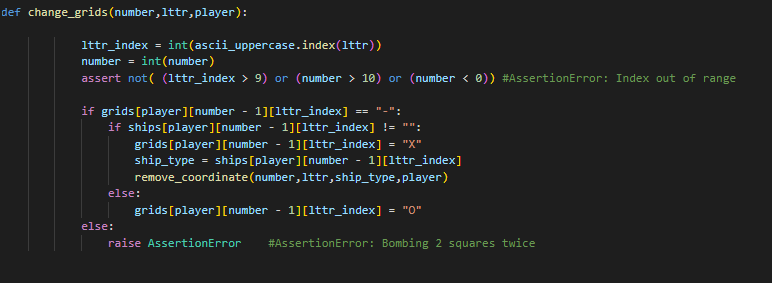


neighbour\_finder



Game\_Table:

It writes every round to the battleship.out and prints to the terminal. It takes the arguments:player and game. Player shows whose move is and game shows the situation “ongoing” and “over”. It changes the outputs according to these informations.



Change\_grids:

It changes the grids which is shown on the game table.If a square has a ship and gets hit that square turns to “X”.If there is no ship square turns to “O”.It raises assertionerror when a square gets hit twice.



Remove\_the\_coordinate:

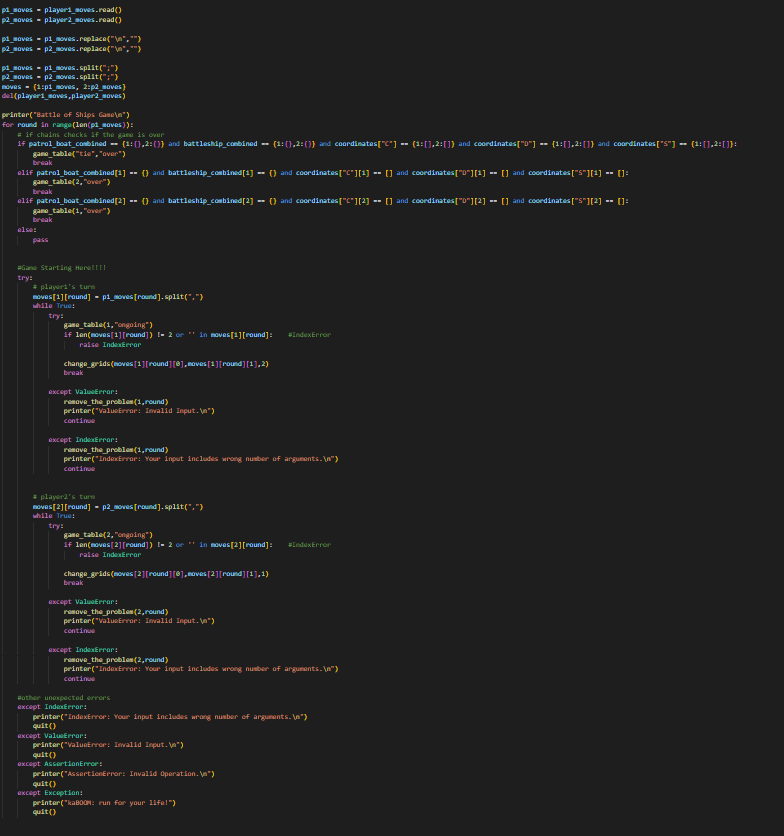
For carries,destroyer and submarine it removes the coordinates which gets hit from the coordinates dictionary. If there is no coordinate left for any ship,ship value turns “X” from the “-“. “X” means that the ship is sunk and it is shown at the game table.For patrol boat it removes the coordinates from patrol\_boat\_combined and then it checks how much there is different values in that dictionary.With that information it changes the values to “X”.It does the same for battleships.



Remove\_the\_problem:

It removes the wrong moves from the moves dictionary.

**Game loop**



Firstly it reads the player1.in and player2.in then strips the “\n”’s and at last split to the semicolons. Then it makes the moves a dictionary for changing them easily. In that for loop game starting round by round. At starting of every round it checks the game is over. If it is not over it starts hitting the grids starting with player1’s move.If there is a wrong move (for example:“,A;” “A;” “,;” “;” “1,;” “1;”) it removes that move from moves and it continues with the same player.

**User Catalogue**

There are 4 different files you need to use :player1.txt player2.txt player1.in and player2.in

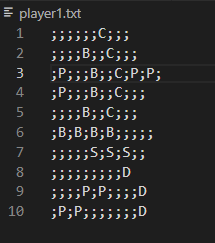
Player1.txt and player2.txt:

It will show the placements of ships.Every line must represent rows and semicolons represents seperations between columns.So every line must contain 9 semicolons. For ship implements you have to use “D” letter for destroyer,”P” letter for patrol boats,”S” letter for submarine,”C” letter for carrier and “B” letter for battleships. Ship sizes and counts must be appropriate to that list:



If you don’t obey to that list you will get an error.

An example of player1.txt:

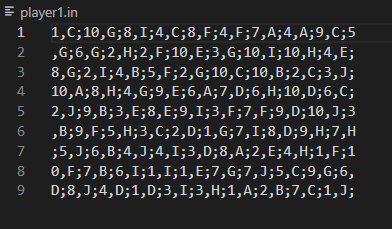


For example first C represents 1,G coordinate.

Player1.in and player2.in:

This text files contains the moves of each players. Moves must be coordinates and it reveals other player’s board.

Player.in’s format must be like this:



First you need to write number which represent row and then the letter which represents the column.Between number and letter you have to use comma or you will get an error. And you have to seperate every move with semicolons otherwise you may get an error. For game rules the count of moves player1 and player2 must be the same or game may not be working properly.

Have fun the game 😊