Hacettepe University
Department of Computer Engineering

Assignment 4: Battle of Ships

2210356024 – Aykut Alp GÜRLER

03/01/2023

Contents

Analysis

Battle of Ships is a strategy game in which players try to sink each other's ships by guessing the locations of their ships on a grid. Each player has a grid with their own ships on it, and they take turns guessing the coordinates of squares on the other player's grid. If the square contains a ship, the player gets a "hit," and if it is empty, they get a "miss." The game continues until one player sinks all of the other player's ships, at which point they win.

In order to play the game, each player needs to place their ships on their grid in secret. The ships can be placed horizontally or vertically, but they cannot be placed diagonally. The ships come in different sizes, with the largest being the "Carrier" which takes up five squares on the grid. Once the ships have been placed, the players take turns guessing the coordinates of squares on the other player's grid. If the square contains a ship, the player gets a hit. If the square is empty, the player gets a miss and it becomes the other player's turn.

The game is won by the player who sinks all of the other player's ships first. Players can try to sink their opponent's ships by using logic and strategy to deduce the locations of the ships based on the hits and misses they have obtained. The game can be played on a physical board with pencil and paper, or it can be played digitally on a computer or mobile device.

In Assignment 4, we do not write a program to play the Battle of Ships game. We write a program of documentation, which is visualized effectively, of played game.

Using multi-dimensional built-in python dictionary as the data structure can enable us to do necessary printing operations.

Assignment 4’s program steps:

- At the beginning of the game, the program reads two files, Player1.txt and Player2.txt, which contain the initial positions of each player's ships. These positions are used to set up the game board for each player. The files are used to determine where the ships are placed on the grid.
- During a player's turn, the program prompts them with the round count, the size of the grid, and two grids representing the current state of the game for each player. The grids show the locations of the ships and whether they have been sunk or are still floating. The opponent's hits are marked with an "X" and their misses are marked with an "O." The cells surrounding the hits and misses that have not been revealed are filled with "-". In the final board, the letters "C", "B", "D", "S", and "P" are used to indicate the locations of the unsunk ships of the winning player that have not been revealed.
- During each round of the game, the players are prompted to enter the coordinates of the cell they want to attack. The coordinates consist of two characters, separated by a comma, representing the row and column index of the cell, respectively. For example, a player might enter "10,D" to attack the cell at row 10 and column D. The program then updates the game board to reflect the attack and the game continues until one player has sunk all of the other player's ships.
- The game continues in this way until it reaches its conclusion. If a player sinks one of their opponent's ships, the program will announce the sinking of the Carrier, Submarine, Destroyer, Patrol Boat, or Battleship. If a player's ships have all been sunk, the game ends and the other player is declared the winner. If both players have sunk all of each other's ships by the end of a round, the game is a draw. The program will then output one of three messages: "Player1 Wins!", "Player2 Wins!", or "It is a Draw!".
- Finally, the program displays the final version of the two players' game boards, with all of the ships and their locations revealed. This allows players to see how the game progressed and how the ships were positioned on the board.

Design

In Assignment 4, I have thought many solutions, functions. In this chapter I am going to explain my code.

Line 3-34

```
def lettoint(letter):
```

Function which takes in letter and returns its corresponding integer value.

Line 36-66

```
def inttolet(integer):
```

Function which takes in integer and returns its corresponding letter string.

Line 68-75

```
def readTXT(txt):
```

Function reads a text file and returns a list of its lines, with leading and trailing white space removed. It does this by opening the file in "r" mode (for reading), which allows you to read the file's contents. The "utf8" encoding ensures that the file is interpreted correctly, regardless of the character set used to encode it.

Line 77-94

```
def shipsPOSITIONS(txt):
```

Function creates a dictionary that stores the positions of each player's ships on a grid. The grid is represented as a list of strings, where each string represents a row on the grid. The function processes each row in the list and creates a dictionary of positions for each player.

The function first initializes an empty dictionary called positionsDict. It then uses a for loop to iterate through each row in the input list, and assigns a row number to the variable row_num.

Within the for loop, the function splits the current row into a list of individual cells using the split() method. It then uses another for loop to iterate through each cell in the row, and assigns a column number to the variable column_num.

For each cell, the function creates a nested dictionary called o_xDict that stores the player's letter ('N', 'B', etc.) as the key, and a dash ('-') as the value. This nested dictionary is then added to the positionsDict dictionary using the cell's coordinates (constructed using the row and column numbers) as the key.

Line 96-107

```
def listPOSITIONS(positionsDict):
```

Function takes a dictionary of positions (positionsDict) and returns a dictionary with lists of positions for each ship. The returned dictionary (shipDict) has keys for each type of ship, and the corresponding values are lists of positions where that ship is located.

The function first initializes an empty dictionary called shipDict, with keys for each type of ship. It then iterates through the keys of the positionsDict dictionary, and for each key (which represents a position on the grid), it iterates through the keys of the nested dictionary at that position.

For each nested dictionary key (which represents a player's letter), the function checks the letter and adds the position to the corresponding list in the shipDict dictionary. For example, if the letter is 'C', the position is added to the list for the "Carrier" ship.

Finally, the function returns the shipDict dictionary.

Line 109-161

```
def finalshipdict(txt,positionsDict):
```

Function takes a list of strings (txt) and a dictionary of positions (positionsDict), and returns a dictionary with lists of positions for each ship. The returned dictionary (shipDict) has keys for each type of ship, and the corresponding values are lists of positions where that ship is located.

The function first calls the listPOSITIONS() function to initialize the shipDict dictionary with the positions of the Carrier, Destroyer, and Submarine ships. It then iterates through the txt list and processes each string in the list to determine the positions of the remaining ships (Battleship1, Battleship2, PatrolBoat1, PatrolBoat2, PatrolBoat3, and PatrolBoat4).

For each string, the function extracts the ship type, starting position, and direction from the string using string manipulation techniques such as split() and indexing. It then uses if statements to determine the correct positions for the ship based on its type and direction, and updates the corresponding value in the shipDict dictionary.

Finally, the function returns the shipDict dictionary.

Line 163-169

```
def playerXmoves(intxt):
```

This function appears to take a list of strings (intxt) and returns a list of moves for each player. The input list is expected to contain strings in the format "1,A;2,B;3,C;4,D;5,E;...", where each string represents a sequence of moves separated by semicolons.

The function processes each string in the input list and concatenates them into a single string using a for loop. It then splits the concatenated string into a list of moves using the split() method and removes the last element of the list using slicing. Finally, the function returns the list of moves.

Line 171-179

```
    def VALFUNC(dictnkey):
```

Function takes a dictionary (dictnkey) and returns the value of the nested dictionary at that key.

The function iterates through the keys of the input dictionary using a for loop and returns the value of the nested dictionary at that key.

Line 181-189

```
    def KEYFUNC(dictnkey):
```

Function takes a dictionary (dictnkey) and returns the key of the nested dictionary at that key.

The function iterates through the keys of the input dictionary using a for loop and returns the key of the nested dictionary at that key.

Line 191-310

```
def processingMoves(positionsDict,shipDict,move,Cinfo,Binfo,Dinfo,Sinfo,Pinfo):
```

Function processes moves in a dictionary of positions (positionsDict) for a game of Battleship. The function takes 8 input arguments: the dictionary of positions, a dictionary of ships, a move (a string representing a position on the game board), and 4 lists containing information about the Carrier, Battleship, Destroyer, and Submarine ships.

The function checks the value of the nested dictionary at the key corresponding to the move in the positionsDict dictionary. If the value is "N", the function updates the value to "O" to indicate a missed shot. If the value is "C", "B", "D", "S", or "P", the function updates the value to "X" to indicate a hit and checks if the respective ship has been sunk by counting the number of "X" values in the ship's positions. If the ship has been sunk, the function updates the respective list with information about the ship's status.

Line 312-320

```
def finalInfo(dictnkey):
```

Function is used to display information about unhit ships in a game of Battleship. The function takes a dictionary (dictnkey) as input and returns either the key of the nested dictionary or the value, depending on the value of the nested dictionary.

The function checks if the value of the nested dictionary is "-". If it is, the function checks if the key of the nested dictionary is not "N". If it is not, the function returns the key of the nested dictionary. Otherwise, the function returns the value of the nested dictionary. If the value of the nested dictionary is not "-", the function returns the value of the nested dictionary.

Line 322-344

```
def printFINALINFO():
```

Code is defining a function printFINALINFO() that will print the final information of a game of battleship. The function prints the board for each player with each cell showing either an 'O' if it has been hit, an 'X' if a ship is placed there, or a letter representing the type of ship if it has not been hit.

The function does this by iterating through each cell on the board and calling finalInfo() on the cell, passing in the value in the positionsDict dictionary for that cell. The finalInfo() function then returns either the value in the nested dictionary for that cell, or the key of the nested dictionary if the value is '-'. This value is then printed as part of the board.

Overall, printFINALINFO() is a function for displaying the final state of the game board at the end of a game of battleship.

Line 346-365

```
def writeFINALINFO():
```

Same explanations for printFINALINFO() is valid for writeFINALINFO() too.

Line 367-424

```
def PRINT_IT(move,turn):
```

Code is printing the current state of the game board. The board is divided into two parts, one for each player. It also includes the round number and grid size. The PRINT_IT function takes two arguments, move and turn. The move argument is not used in the function. The turn argument is a boolean value that indicates whose turn it is. The function then uses this value to determine whether to print "Player1's Move" or "Player2's Move".

The function then prints the game board by looping through the positionsDict1 and positionsDict2 dictionaries and using the VALFUNC function to get the value associated with each key. It prints the value of the positionsDict1 dictionary on the left side of the board and the value of the positionsDict2 dictionary on the right side of the board.

Finally, the function writes the same output to a text file using the f.write function.

Line 426-483

```
def MovesLikeJagger():
```

The function MovesLikeJagger() is the main function of the program. It is responsible for processing a series of moves by two players in a game of battleship. It keeps track of which player's turn it is using a boolean variable first_turn, and processes moves from the player1moves and player2moves lists by alternating between the two.

Before processing a move, the function checks the move for validity. It does this by splitting the move string on the comma and checking that the resulting list has exactly two elements, that neither element is an empty string, and that the first element is a valid integer between 1 and 10, inclusive, and the second element is a valid column letter between 'A' and 'J', inclusive. If the move is not valid, an error message is printed and the function moves on to the next move.

If the move is valid, the function removes the comma from the move string, calls the PRINT_IT() function with the move and the value of first_turn, and then flips the value of first_turn to alternate between players.

MovesLikeJagger() will keep running until it has processed all of the moves in the player1moves and player2moves lists.

Line 486

```
with open("Battleship.out", "w") as f:
```

Code is opening a file called "Battleship.out" in write mode, and assigning the file object to the variable f. This will allow you to write to the file using the f.write() method.

Line 487-519 and 557-562:

A problem can occur if the input files are not reachable at the specified file path (if files do not exist, or if their names are misspelled, etc..). This block handles exceptions by printing “IOError: input file(s) Player1.txt is/are not reachable.” or “IOError: inputfile(s) Player1.txt, Player2.txt, Player1.in, Player2.in is/are not reachable.”.

Line 523-537:

This block of code for checking ship infos.

Line 542-556:

This code block is checking if any of the players has won the game. The variables Cinfo1, Binfo1, Dinfo1, Sinfo1, and Pinfo1 are the number of ships of each type that player 1 has left. The same goes for Cinfo2, Binfo2, Dinfo2, Sinfo2, and Pinfo2. The variables Cinfocheck, Binfocheck, Dinfocheck, Sinfocheck, and Pinfocheck are the total number of ships of each type that each player has at the start of the game.

If the number of ships of each type that each player has left is equal to the number of ships of each type that each player had at the start of the game, it means that it is a draw. Otherwise, if player 1 has no ships left, player 2 wins. If player 2 has no ships left, player 1 wins.

Programmer’s Catalogue

- For analyzing, I have spent approximately 6 hours initially. Totally, analyzing takes 10+ hours.
- For designing, I have spent approximately 1 days.
- For implementing, I have spent approximately 5-6 days.
- For testing, I have spent approximately 3-4 hours.
- For reporting, I have spent approximately 7-8 hours.
- I have totally used 15 functions, I have tried to use as less global variables as I can. So I think my program is reusable by other programmers because my code is easy to understand.
- I have explained my code in detail in chapter “Design”. Take a look at there.

User Catalogue

1. For documentation, Firstly, create “Player1.txt”, “Player2.txt” files with player’s ship positions.
2. Secondly, you should be tracked moves played by Player1 and Player2. Create “Player1.in”, “Player2.in” files and write moves in them.
3. Now, all data program needs is ready!
4. Execute the program and see visualized “Battle of Ship” game documentation in “Battleship.out” file.
5. You can analyze game with its all rounds, even with “Final Information”!!! ☺