Hacettepe University

Department Of Artificial Intelligence

# BBM103 Assignment 4 Report

# Creating the Game “Battleship”

Adam Sattout - b2220765061

06/1/2023

# 1 Analysis

Here I will explain the game we are going to make and explain the outcomes needed.

In this assignment we have to recreate the well-known game “Battleship”. This game can be seen as a guessing game between 2 players. Each player has a hidden grid that he can set up his ships with their different types. Each type has a specific length, a carrier is 5 blocks long, a battleship is 4, a submarine and a destroyer are 3, and the patrol ships are 2 blocks long. Both players will take turns shooting the hidden board and the program will show if that shot hit a ship or hit nothing, with keeping count of ships of the players and their types. The loop continues until one of the players loses all his ships or both losing their ships resulting in a draw.

# 2 Design

In this section I will talk about the program’s main working method that I used in this project.

For this game we first take the placements of the ships of both players in txt files and create a hidden grid according to those files, then it defines the ships of the players and sorts them with their type into a dictionary, after that the game starts with taking guesses from players and calling the right functions to print the output of guesses and defining the game’s result, whether one or both players won or the game isn’t done yet. If the game is over the program will print the final information with announcing the winner.

## 2.1 Implementation

For this section, I will furthermore expand upon each function I have created in this program before I provide the whole code in the programmer catalog section.

### 2.1.1 Reading the Input Files

```
def fileListReturner(file, splitter):
    try:
        input = open(file)
    except IOError:
        return file
    inputList = input.read().split(splitter)
    input.close()
    return inputList

iostring = ""
try:
    p1InputList = fileListReturner(sys.argv[1], "\n")
    p2InputList = fileListReturner(sys.argv[2], "\n")
    p1Guess = fileListReturner(sys.argv[3], ";")
    p2Guess = fileListReturner(sys.argv[4], ";")
except:
    print("Less arguments were entered than needed")
    oF.write("Less arguments were entered than needed")
    oF.close()
    exit()

if isinstance(p1InputList, str) == True:
   iostring += p1InputList + ", "
if isinstance(p2InputList, str) == True:
   iostring += p2InputList + ", "
if isinstance(p1Guess, str) == True:
   iostring += p1Guess + ", "
if isinstance(p2Guess, str) == True:
    iostring += p2Guess + ", "

if iostring != "":
    iostring = iostring[:-2]
    print("IOError: inputfile(s) %s is/are not reachable."%iostring)
    oF.write("IOError: inputfile(s) %s is/are not reachable."%iostring)
    oF.close()
    exit()
```

After we create the heading of the main dictionaries of the players, we define this function that starts by identifying all output files (guesses and ship placements), then it opens the file specified as an argument, reads it, and splits its content into a list. If the program doesn't find the file it will invoke the IOError exception and return a string with the called file's name to be able to print a message to the users informing them with this error. We call the function with the 4 arguments of the user as names of the files that have the inputs, if the arguments were lower than expected the program will raise an indexerror informing the user as well.

```
Try to open file with given name
Except IOError return filename string
Create a list with content of the file split by the specified splitter and return it
Try to assign variables to the returning list of the fileListReturner function
Except inform user there is an error
```

### 2.1.2 Creating Players' Dictionaries

```
def dictCreate(dictName, characterList, inputList):
    gridList = []
    placementCout = 0
    for i in range(10):
        formattedList = inputList[i].split(";")
        assert len(formattedList) == 10
        charIndex = -1
        for j in characterList[:10]:
            charIndex += 1
            keyName = j + str(i+1)
            if formattedList[charIndex] == "":
                dictName[keyName] = "-"
            elif formattedList[charIndex] == "C":
                dictName[keyName] = "C"
                placementCout += 1
            elif formattedList[charIndex] == "B":
                dictName[keyName] = "B"
                placementCout += 1
            elif formattedList[charIndex] == "P":
                dictName[keyName] = "P"
                placementCout += 1
            elif formattedList[charIndex] == "D":
                dictName[keyName] = "D"
                placementCout += 1
            elif formattedList[charIndex] == "S":
                dictName[keyName] = "S"
                placementCout += 1
            else:
                raise AssertionError
            gridList.append(keyName)
    assert placementCout == 27
    return dictName, gridList

try:
    p1Dict, gridList = dictCreate(p1Dict, characterList, p1InputList)
    p2Dict, gridList = dictCreate(p2Dict, characterList, p2InputList)
except:
    oF.write("kaBOOM: run for your life!\n")
    print("kaBOOM: run for your life!")
```

To be able to record players ship placements, we first take the input of each user line by line and check in the input if a space is empty or has a ship type labeled, the function will create a grid with keys A1, B1, C1… to have a 10x10 grid, if it encounters an empty space the function will fill the key value with a simple “-”. Otherwise, the function will fill it with a letter referring to the ship type. For instance, a “c” refers to a Carrier while a “P” refers to a Patrol ship, and the program will check if more than the allowed letters were inserted by asserting there are only 27 letters.

Take line of input file
Assert its length is 10

```
For each object in the line:
        Create a grid piece as a key for the player's grid
        Evaluate what the value of that key should be based on input
Return the grid created
If AssertionError occurs:
        Inform the user
```

## 2.1.3 Defining the Ships of the Players

```
def shipDefiner(dict, gridList):
    shipDict = {}
    tempDict = {}
    tempDict = dict
    shipCout = 0
    subCout = 0
    for key, value in tempDict.items():
        indexIdentifier = gridList.index(key)
        if value == "-":
            continue
        if tempDict[gridList[indexIdentifier + 1]] == value:
            if value == "P":
                string = "Patrol" + key
                shipDict[string] = key + "-" + gridList[indexIdentifier + 1]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 1]] == "P"
                tempDict[gridList[indexIdentifier + 1]] = "-"
            elif value == "S":
                string = "Submarine" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 2]
                shipCout += 1
                subCout += 1
                assert  tempDict[gridList[indexIdentifier + 1]] == "S" and  tempDict[gridList[indexIdentifier + 2]] == "S"
                tempDict[gridList[indexIdentifier + 1]] = "-"
                tempDict[gridList[indexIdentifier + 2]] = "-"
            elif value == "D":
                string = "Destroyer" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 2]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 1]] == "D" and  tempDict[gridList[indexIdentifier + 2]] == "D"
                tempDict[gridList[indexIdentifier + 1]] = "-"
                tempDict[gridList[indexIdentifier + 2]] = "-"
            elif value == "B":
                string = "Battleship" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 3]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 1]] == "B" and  tempDict[gridList[indexIdentifier + 2]] == "B" and
tempDict[gridList[indexIdentifier + 3]] == "B"
                tempDict[gridList[indexIdentifier + 1]] = "-"
                tempDict[gridList[indexIdentifier + 2]] = "-"
                tempDict[gridList[indexIdentifier + 3]] = "-"
            elif value == "C":
                string = "Carrier" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 4]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 1]] == "C" and  tempDict[gridList[indexIdentifier + 2]] == "C"
                assert  tempDict[gridList[indexIdentifier + 3]] == "C" and  tempDict[gridList[indexIdentifier + 4]] == "C"
                tempDict[gridList[indexIdentifier + 1]] = "-"
                tempDict[gridList[indexIdentifier + 2]] = "-"
                tempDict[gridList[indexIdentifier + 3]] = "-"
                tempDict[gridList[indexIdentifier + 4]] = "-"
        else:
            if value == "P":
                string = "Patrol" + key
                shipDict[string] = key + "-" + gridList[indexIdentifier + 10]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 10]] == "P"
                tempDict[gridList[indexIdentifier + 10]] = "-"
            elif value == "S":
                string = "Submarine" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 20]
                shipCout += 1
                subCout += 1
                assert  tempDict[gridList[indexIdentifier + 10]] == "S" and  tempDict[gridList[indexIdentifier + 20]] == "S"
                tempDict[gridList[indexIdentifier + 10]] = "-"
                tempDict[gridList[indexIdentifier + 20]] = "-"
            elif value == "D":
                string = "Destroyer" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 20]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 10]] == "D" and  tempDict[gridList[indexIdentifier + 20]] == "D"
                tempDict[gridList[indexIdentifier + 10]] = "-"
                tempDict[gridList[indexIdentifier + 20]] = "-"
            elif value == "B":
                string = "Battleship" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 30]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 10]] == "B" and  tempDict[gridList[indexIdentifier + 20]] == "B" and
tempDict[gridList[indexIdentifier + 30]] == "B"
                tempDict[gridList[indexIdentifier + 10]] = "-"
                tempDict[gridList[indexIdentifier + 20]] = "-"
                tempDict[gridList[indexIdentifier + 30]] = "-"
            elif value == "C":
                string = "Carrier" + key
                shipDict[string] = key + "-" + gridList[gridList.index(key) + 40]
                shipCout += 1
                assert  tempDict[gridList[indexIdentifier + 10]] == "C" and  tempDict[gridList[indexIdentifier + 20]] == "C"
                assert  tempDict[gridList[indexIdentifier + 30]] == "C" and  tempDict[gridList[indexIdentifier + 40]] == "C"
                tempDict[gridList[indexIdentifier + 10]] = "-"
                tempDict[gridList[indexIdentifier + 20]] = "-"
                tempDict[gridList[indexIdentifier + 30]] = "-"
                tempDict[gridList[indexIdentifier + 40]] = "-"
    assert shipCout == 9
    assert subCout == 1
    return shipDict
```

This part of the code is really important to define the placements of the ships so the program can decide where the ships were placed and identify if a ship was hit or not later on. The function starts of by looping through the player’s grid. If it encounters a piece that has a ship on it, the function will identify what type of ship that is, identify whether the ship was placed horizontally or vertically, assert that it was placed with the right length, and record the ship into a dictionary that holds players' ships locations, after that it will change the values of the remainder of the ship to blanks so the program doesn't record the ship that was already recorded twice. It also counts how many ships are created to mimic the rules of the main game having a limit of how many ships a player can place. Finally, it returns the dictionary of the ships of the player.

```
Loop in player's grid
    If value != blank
        Assert ship length and type are matched
        Change rest of ship to blank
        Record ship
        ShipCount += 1
Assert ShipCount is right
Return ships recorded
```

### 2.1.4 Checking Ships Statuses

```python
def gameoverChecker(p1Ships, p1Table, characterList, gridList):
    patrolsDead = 0
    submarinesDead = 0
    destroyersDead = 0
    battleshipsDead = 0
    carriersDead = 0
    for key, value in p1Ships.items():
        checkingRange = value.split("-")
        if gridList.index(checkingRange[1]) - gridList.index(checkingRange[0]) < 10:
            shot = 0
            fromm = gridList.index(checkingRange[0])
            to = gridList.index(checkingRange[1])
            while fromm <= to:
                if p1Table[gridList[fromm]] == "X":
                    shot += 1
                fromm += 1
            if "Patrol" in key and shot == 2:
                patrolsDead += 1
            elif "Submarine" in key and shot == 3:
                submarinesDead += 1
            elif "Destroyer" in key and shot == 3:
                destroyersDead += 1
            elif "Battleship" in key and shot == 4:
                battleshipsDead += 1
            elif "Carrier" in key and shot == 5:
                carriersDead += 1
        else:
            shot = 0
            for i in range(int(checkingRange[0][1:]), int(checkingRange[1][1:]) + 1):
                checkingKey = value[0] + str(i)
                if p1Table[checkingKey] == "X":
                    shot += 1
            if "Patrol" in key and shot == 2:
                patrolsDead += 1
            elif "Submarine" in key and shot == 3:
                submarinesDead += 1
            elif "Destroyer" in key and shot == 3:
                destroyersDead += 1
            elif "Battleship" in key and shot == 4:
                battleshipsDead += 1
            elif "Carrier" in key and shot == 5:
                carriersDead += 1

    return patrolsDead, submarinesDead, destroyersDead, battleshipsDead, carriersDead
```

In this part of the code, the program will check for the number of ships destroyed when it is called. It loops through the dictionary of the defined ships of the player and checks for that ship with its coordinates as values on the player’s visible grid (that records the shots of the other player). If all the pieces of the ship are shot, the program will raise the number of ships of that type destroyed by 1. At the end, the dunction will return the number of ships destroyed for all types.

```
For item in ships dictionary
        Check the location associated with the ship label
        If It was all shot
                Increase number of dead ships of that type
Return number of ships dead of each type
```

## 2.1.5 Guessing Input’s Error Checking

```
def errorCheck(moveIndex, characterList, gridList, guessList):
    if moveIndex == len(guessList):
        return "needGuess", "NG", False
    try:
        sit = 0
        errorAv = False
        errorString = "\n"
        move = guessList[moveIndex].split(",")
        y = move[0]
        x = move[1]
        if x == "" or y == "":
            sit = 1
            raise IndexError
        str(int(y))
        keyName = x + y
        if len(move) > 2 or x not in characterList:
            raise ValueError
        assert keyName in gridList
    except ValueError:
        errorString = "ValueError: Couldnt process the value entered into the grid"
        errorAv = True

    except IndexError:
        if sit == 0:
            errorString = "IndexError: Entered less values than needed"
        elif sit == 1:
            errorString = "IndexError: Entered an empty value"
        errorAv = True
    except AssertionError:
        errorString = "AssertionError: Invalid Operation."
        errorAv = True
    except:
        errorString = "kaBOOM: run for your life!"
        errorAv = True
    else:
        return errorString, keyName, errorAv
    return errorString, "", errorAv
```

To be able to check if there is an error in a player's move we compare it with the main format of [letter],[number] with the move being available in the main grid. It first tries to see if the move isn't even available due to lack of guesses before the game ending, then it tries to see if there are any lack of input in the move, like typing a letter only or nothing at all. After that it checks if the number that was given is a real number and if the letter given is a real letter, if the move passes that as well, the function will check if the move is in the grid. If it is really in the grid, the function will return the move with the variable ErrorAv equal to False indicating that there are no errors. However, if any of these errors were encountered the function will return the appropriate string to print according to the error type and return the ErrorAv variable equal to True.

```
Check if move has empty inputs
Check if both x and y coordinates are appropriate values
Check if more than 2 coordinates were given
Check if move is in gridlist
If there is no errors
        Return move and ErrorAv = False
Else
        Return error string and ErrorAv = True
```

## 2.1.6 Printing the User Interface

```
def tablePrinter(characterList,gridList,table, table2, caller):
  p1Win = False
  p2Win = False
  gameOver = False
  pat2, sub2, dest2, batt2, car2 = gameoverChecker(p2Ships, p2Table, characterList, gridList)
  pat1, sub1, dest1, batt1, car1 = gameoverChecker(p1Ships, p1Table, characterList, gridList)

  if caller == "checker":
    if pat1 == 4 and sub1 == 1 and dest1 == 1 and batt1 == 2 and car1 == 1:
      gameOver = True
      p2Win = True
      if pat2 == 4 and sub2 == 1 and dest2 == 1 and batt2 == 2 and car2 == 1:
        p1Win = True
    elif pat2 == 4 and sub2 == 1 and dest2 == 1 and batt2 == 2 and car2 == 1:
      gameOver = True
      p1Win = True
    return gameOver, p1Win, p2Win

  if caller == "printer":
    oF.write("%s\t\t%s"%("Player1's Hidden Board", "Player2's Hidden Board\n  "))
    print("%s\t\t%s"%("Player1's Hidden Board", "Player2's Hidden Board\n  "), end = "")
  elif caller == "printer2":
    oF.write("%s\t\t\t\t%s"%("Player1's Board", "Player2's Board\n  "))
    print("%s\t\t\t%s"%("Player1's Board", "Player2's Board\n  "), end = "")

  for i in characterList[:characterList.index(gridList[-1][0]) + 1]:
    oF.write("%-2s"%i)
    print("%-2s"%i, end = "")
  oF.write("\t\t  ")
  print("\t\t  ", end = "")

  for i in characterList[:characterList.index(gridList[-1][0]) + 1]:
    if i != "J":
      oF.write("%-2s"%i)
      print("%-2s"%i, end = "")
    else:
      oF.write(("%-2s"%i)[:-1])
      print(("%-2s"%i)[:-1], end = "")
  print()
  oF.write("\n")

  for i in range(1, int(gridList[-1][1:]) + 1):
    oF.write("%-2d"%i)
    print("%-2d"%i, end = "")

    for j in characterList[:characterList.index(gridList[-1][0]) + 1]:
        piece = j + str(i)
        oF.write("%-2s"%table[piece])
        print("%-2s"%table[piece], end = "")
    oF.write("\t\t%-2d"%i)
    print("\t\t%-2d"%i, end = "")

    for j in characterList[:characterList.index(gridList[-1][0]) + 1]:
        piece = j + str(i)
        if j != "J":
            oF.write("%-2s"%table2[piece])
            print("%-2s"%table2[piece], end = "")
        else:
            oF.write(("%-2s"%table2[piece])[:-1])
            print(("%-2s"%table2[piece])[:-1], end = "")
    oF.write("\n")
    print()

print("\nCarrier\t\t%s\t\tCarrier\t\t%s"%(("X "*car1 + "- "*(1-car1))[:-1], ("X "*car2 + "- "*(1-car2))[:-1]), end = "")
print("\nBattleship\t%s\t\tBattleship\t%s"%(("X "*batt1 + "- "*(2-batt1))[:-1], ("X "*batt2 + "- "*(2-batt2))[:-1]), end = "")
print("\nDestroyer\t%s\t\tDestroyer\t%s"%(("X "*dest1 + "- "*(1-dest1))[:-1], ("X "*dest2 + "- "*(1-dest2))[:-1]), end = "")
print("\nSubmarine\t%s\t\tSubmarine\t%s"%((("X "*sub1 + "- "*(1-sub1)))[:-1], ("X "*sub2 + "- "*(1-sub2))[:-1]), end = "")
print("\nPatrol Boat\t%s\t\tPatrol Boat\t%s"%(("X "*pat1 + "- "*(4-pat1))[:-1], ("X "*pat2 + "- "*(4-pat2))[:-1]))
oF.write("\nCarrier\t\t%s\t\t\t\tCarrier\t\t%s"%((("X "*car1 + "- "*(1-car1))[:-1], ("X "*car2 + "- "*(1-car2))[:-1])))
oF.write("\nBattleship\t%s\t\t\t\tBattleship\t%s"%(("X "*batt1 + "- "*(2-batt1))[:-1], ("X "*batt2 + "- "*(2-batt2))[:-1]))
oF.write("\nDestroyer\t%s\t\t\t\tDestroyer\t%s"%(("X "*dest1 + "- "*(1-dest1))[:-1], ("X "*dest2 + "- "*(1-dest2))[:-1]))
oF.write("\nSubmarine\t%s\t\t\t\tSubmarine\t%s"%(("X "*sub1 + "- "*(1-sub1))[:-1], ("X "*sub2 + "- "*(1-sub2))[:-1]))
oF.write("\nPatrol Boat\t%s\t\t\tPatrol Boat\t%s\n"%(("X "*pat1 + "- "*(4-pat1))[:-1], ("X "*pat2 + "- "*(4-pat2))[:-1]))
if caller != "printer2":
    oF.write("\n")
```

This function is the one that will print all of the game’s events to the players when called. It can be called by 3 callers, “checker” (which will return whether player 1 won, player 2 won, or both of them won based on the gameoverChecker returned values), “printer” (which will print the game’s development at a certain move in a strict way so no extra spaces or tabs are added), and lastly “printer2” (which is another version of the main printer but it is only called at the end to remove an extra line to match the Battleship.out file given with this assignment EXACTLY)

```
Check for the caller’s type
If caller is checker
    Check if the game is over
If caller is printer
    Print the game update at the specified move
```

If caller is printer2

Print the game update at the specified move with the last move format

## 2.1.6 Processing the Main Game Loop

```
while gameOver == False:
    round += 1
    if moveIndex1 == len(p1Guess) or moveIndex2 == len(p2Guess):
        break

    errorString1, keyName1, errorAv1 = errorCheck(moveIndex1, characterList, gridList, p1Guess)
    print("\nPlayer1's Move\n\nRound : %d\t\t\t\tGrid Size: %dx%d\n\n"%(round,
characterList.index(gridList[-1][0]) + 1, int(gridList[-1][1:])), end = "")
    oF.write("\nPlayer1's Move\n\nRound : %d\t\t\t\tGrid Size: %dx%d\n\n"%(round,
characterList.index(gridList[-1][0]) + 1, int(gridList[-1][1:])))
    tablePrinter(characterList, gridList, p1Table, p2Table, "printer")
    oF.write("Enter your move: %s\n"%p1Guess[moveIndex1])
    print("\nEnter your move: %s"%p1Guess[moveIndex1])

    while errorAv1 == True:
        oF.write(errorString1)
        print(errorString1, end = "")
        moveIndex1 += 1
        if moveIndex1 < len(p1Guess):
            moveIn = p1Guess[moveIndex1]
        else:
            moveIn = ""
        oF.write("\nEnter your move: %s\n"%moveIn)
        print("\nEnter your move: %s\n"%moveIn)
        errorString1, keyName1, errorAv1 = errorCheck(moveIndex1, characterList, gridList, p1Guess)


    if keyName1 == "NG":
        break

    if p2Dict[keyName1] != "-":
        p2Table[keyName1] = "X"
        p2Dict[keyName1] = "X"
    else:
        p2Table[keyName1] = "O"
        p2Dict[keyName1] = "O"
    moveIndex1 += 1
```

This part of code can be considered the main part of the code where everything is processed and called. It processes the moves one by one and modifies player's main dictionaries and their visible tables. It also calls the function errorChecker() to check if the input was wrong or not, as well as printing some additions to the user interface before and after calling the function tablePrinter()

```
Check if the move is correct and if guesses aren't enough
Print the tables corresponding to the present move
Call the error checking function
If there is an error
    Inform the user
    Move to the next move and check again until a right one is inserted
Do the same for player 2
```

## 2.1.7 Checking If the Game Is Over

```
if gameOver == True and p1Win == True and p2Win == False:
   oF.write("\nPlayer1 Wins!\n\n")
   print("\nPlayer1 Wins!\n")
   p1Table = p1Dict
elif gameOver == True and p2Win == True and p1Win == False:
   oF.write("\nPlayer2 Wins!\n\n")
   print("\nPlayer2 Wins!\n")
   p2Table = p2Dict
elif gameOver == True and p2Win == True and p1Win == True:
   print('\nIt is a Draw!\n')
   oF.write("\nIt is a Draw!\n\n")
if gameOver == True:
   oF.write("Final Information\n\n")
   print("Final Information\n")
   tablePrinter(characterList,gridList,p1Table, p2Table, "printer2")
```

This is the final part of the code that checks whether the game is over or not and who the winner is to print the appropriate result with the help of the tablePrinter() function.

```
If the game is over and player 1 won
    Inform the players that player 1 won
If the game is over and player 2 won
    Inform the players that player 2 won
If the game is over and both players won
    Inform the players it is a draw
```

```
If the game is over
        Print the final information of the game with the hidden dictionaries visible
```

## 3 Programmer’s Catalogue

In this section I will provide the code I’ve written as well as some information about the time spent making this assignment

#We start off by importing the libraries we are going to need and defining dicts and characterlist containing all English letters that we are going to use
#through out the code and create the main outputFile

```
import string
import sys
characterList = list(string.ascii_uppercase)
p1Dict = {}
p2Dict = {}
oF = open("Battleship.out", "w")
```

#Then we take arguments from the cmd and enter them into this function to return us a list splitted by the needed spliter depending on the file format, and
#if there is an error we return a string with the inserted argument and check if any string was returned we print a string calling out the file
#called wasnt found

```
def fileListReturner(file, splitter):
    try:
        input = open(file)
    except IOError:
        return file
    inputList = input.read().split(splitter)
    input.close()
    return inputList

iostring = ""
try:
    p1InputList = fileListReturner(sys.argv[1], "\n")
    p2InputList = fileListReturner(sys.argv[2], "\n")
    p1Guess = fileListReturner(sys.argv[3], ";")
    p2Guess = fileListReturner(sys.argv[4], ";")
except:
    print("Less arguments were entered than needed")
    oF.write("Less arguments were entered than needed")
    oF.close()
    exit()

if isinstance(p1InputList, str) == True:
   iostring += p1InputList + ", "
```

```
if isinstance(p2InputList, str) == True:
  iostring += p2InputList + ", "
if isinstance(p1Guess, str) == True:
  iostring += p1Guess + ", "
if isinstance(p2Guess, str) == True:
   iostring += p2Guess + ", "

if iostring != "":
   iostring = iostring[:-2]
   print("IOError: inputfile(s) %s is/are not reachable."%iostring)
   oF.write("IOError: inputfile(s) %s is/are not reachable."%iostring)
   oF.close()
   exit()
```

#This function creates the main boards of each player after the inputs of ships are places in the txt files, after we split the file with the previous function
#we split it once more with the splitter ; to take each column's input and add it to the grid of the player with an index on the matrix according to its
#location, asserting that there are 10 columns and no random inputs were given, and returning both player's grid and a grid we compare it with and using it
#for the rest of the program

```
def dictCreate(dictName, characterList, inputList):
   gridList = []
   placementCout = 0
   for i in range(10):
      formattedList = inputList[i].split(";")
      assert len(formattedList) == 10
      charIndex = -1
      for j in characterList[:10]:
         charIndex += 1
         keyName = j + str(i+1)
         if formattedList[charIndex] == "":
            dictName[keyName] = "-"
         elif formattedList[charIndex] == "C":
            dictName[keyName] = "C"
            placementCout += 1
         elif formattedList[charIndex] == "B":
            dictName[keyName] = "B"
            placementCout += 1
         elif formattedList[charIndex] == "P":
            dictName[keyName] = "P"
            placementCout += 1
         elif formattedList[charIndex] == "D":
            dictName[keyName] = "D"
            placementCout += 1
         elif formattedList[charIndex] == "S":
            dictName[keyName] = "S"
            placementCout += 1
         else:
            raise AssertionError
         gridList.append(keyName)
   assert placementCout == 27
   return dictName, gridList

try:
   p1Dict, gridList = dictCreate(p1Dict, characterList, p1InputList)
   p2Dict, gridList = dictCreate(p2Dict, characterList, p2InputList)
except:
   oF.write("kaBOOM: run for your life!\n")
   print("kaBOOM: run for your life!")
```

#After that comes defining the ships according to players' grids and putting them in a dict with the ship type and its location range, it calculates how long the ship
#is and its location based on the first occurance of a letter and the length of the ship that belongs to that letter, after looping through out the whole
#grid dthe function returns the player's shipdict and asserts that ships' lengths are right as well as that ship types' number is correct

```
def shipDefiner(dict, gridList):
    shipDict = {}
    tempDict = {}
    tempDict = dict
    shipCout = 0
    subCout = 0
    for key, value in tempDict.items():
      indexIdentifier = gridList.index(key)
      if value == "-":
        continue
      if tempDict[gridList[indexIdentifier + 1]] == value:
        if value == "P":
          string = "Patrol" + key
          shipDict[string] = key + "-" + gridList[indexIdentifier + 1]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 1]] == "P"
          tempDict[gridList[indexIdentifier + 1]] = "-"
        elif value == "S":
          string = "Submarine" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 2]
          shipCout += 1
          subCout += 1
          assert  tempDict[gridList[indexIdentifier + 1]] == "S" and  tempDict[gridList[indexIdentifier + 2]]
== "S"
          tempDict[gridList[indexIdentifier + 1]] = "-"
          tempDict[gridList[indexIdentifier + 2]] = "-"
        elif value == "D":
          string = "Destroyer" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 2]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 1]] == "D" and  tempDict[gridList[indexIdentifier + 2]]
== "D"
          tempDict[gridList[indexIdentifier + 1]] = "-"
          tempDict[gridList[indexIdentifier + 2]] = "-"
        elif value == "B":
          string = "Battleship" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 3]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 1]] == "B" and  tempDict[gridList[indexIdentifier + 2]]
== "B" and tempDict[gridList[indexIdentifier + 3]] == "B"
          tempDict[gridList[indexIdentifier + 1]] = "-"
          tempDict[gridList[indexIdentifier + 2]] = "-"
          tempDict[gridList[indexIdentifier + 3]] = "-"
        elif value == "C":
          string = "Carrier" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 4]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 1]] == "C" and  tempDict[gridList[indexIdentifier + 2]]
== "C"
          assert  tempDict[gridList[indexIdentifier + 3]] == "C" and  tempDict[gridList[indexIdentifier + 4]]
== "C"
          tempDict[gridList[indexIdentifier + 1]] = "-"
          tempDict[gridList[indexIdentifier + 2]] = "-"
          tempDict[gridList[indexIdentifier + 3]] = "-"
          tempDict[gridList[indexIdentifier + 4]] = "-"
      else:
        if value == "P":
          string = "Patrol" + key
          shipDict[string] = key + "-" + gridList[indexIdentifier + 10]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 10]] == "P"
          tempDict[gridList[indexIdentifier + 10]] = "-"
        elif value == "S":
          string = "Submarine" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 20]
          shipCout += 1
          subCout += 1
          assert  tempDict[gridList[indexIdentifier + 10]] == "S" and  tempDict[gridList[indexIdentifier +
20]] == "S"
          tempDict[gridList[indexIdentifier + 10]] = "-"
          tempDict[gridList[indexIdentifier + 20]] = "-"
        elif value == "D":
          string = "Destroyer" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 20]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 10]] == "D" and  tempDict[gridList[indexIdentifier +
20]] == "D"
          tempDict[gridList[indexIdentifier + 10]] = "-"
          tempDict[gridList[indexIdentifier + 20]] = "-"
        elif value == "B":
          string = "Battleship" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 30]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 10]] == "B" and  tempDict[gridList[indexIdentifier +
20]] == "B" and tempDict[gridList[indexIdentifier + 30]] == "B"
          tempDict[gridList[indexIdentifier + 10]] = "-"
          tempDict[gridList[indexIdentifier + 20]] = "-"
          tempDict[gridList[indexIdentifier + 30]] = "-"
        elif value == "C":
          string = "Carrier" + key
          shipDict[string] = key + "-" + gridList[gridList.index(key) + 40]
          shipCout += 1
          assert  tempDict[gridList[indexIdentifier + 10]] == "C" and  tempDict[gridList[indexIdentifier +
20]] == "C"
          assert  tempDict[gridList[indexIdentifier + 30]] == "C" and  tempDict[gridList[indexIdentifier +
40]] == "C"
          tempDict[gridList[indexIdentifier + 10]] = "-"
          tempDict[gridList[indexIdentifier + 20]] = "-"
          tempDict[gridList[indexIdentifier + 30]] = "-"
          tempDict[gridList[indexIdentifier + 40]] = "-"
assert shipCout == 9
assert subCout == 1
return shipDict

#This is a small function that creates the visible grids for the players
def hiddenTablesCreate(gridList):
table = {}
for i in gridList:
      table[i] = "-"
return table

try:
p1Ships = shipDefiner(p1Dict, gridList)
p2Ships = shipDefiner(p2Dict, gridList)
except:
oF.write("kaBOOM: run for your life!\n")
print("kaBOOM: run for your life!")


p1Table = hiddenTablesCreate(gridList)
p2Table = hiddenTablesCreate(gridList)

p1Dict, gridList = dictCreate(p1Dict, characterList, p1InputList)
```

```
p2Dict, gridList = dictCreate(p2Dict, characterList, p2InputList)

#last input is empty so we remove it

p1Guess.remove(p1Guess[-1])
p2Guess.remove(p2Guess[-1])

#for game checking we first check players' ships locations and take the range ships are into account, then compare them with the visible tables being shot at that
#location. which += 1 the shot count, if shot count meets the length of the ship type taken from the ships dict the function will return that a ship of that
#type was destroyed (horizontal and vertical ships are taken into account in different ways which is why we use if twice)
def gameoverChecker(p1Ships, p1Table, characterList, gridList):
  patrolsDead = 0
  submarinesDead = 0
  destroyersDead = 0
  battleshipsDead = 0
  carriersDead = 0
  for key, value in p1Ships.items():
    checkingRange = value.split("-")
    if gridList.index(checkingRange[1]) - gridList.index(checkingRange[0]) < 10:
      shot = 0
      fromm = gridList.index(checkingRange[0])
      to = gridList.index(checkingRange[1])
      while fromm <= to:
        if p1Table[gridList[fromm]] == "X":
          shot += 1
        fromm += 1
      if "Patrol" in key and shot == 2:
        patrolsDead += 1
      elif "Submarine" in key and shot == 3:
        submarinesDead += 1
      elif "Destroyer" in key and shot == 3:
        destroyersDead += 1
      elif "Battleship" in key and shot == 4:
        battleshipsDead += 1
      elif "Carrier" in key and shot == 5:
        carriersDead += 1
    else:
      shot = 0
      for i in range(int(checkingRange[0][1:]), int(checkingRange[1][1:]) + 1):
        checkingKey = value[0] + str(i)
        if p1Table[checkingKey] == "X":
          shot += 1
      if "Patrol" in key and shot == 2:
        patrolsDead += 1
      elif "Submarine" in key and shot == 3:
            submarinesDead += 1
        elif "Destroyer" in key and shot == 3:
            destroyersDead += 1
        elif "Battleship" in key and shot == 4:
            battleshipsDead += 1
        elif "Carrier" in key and shot == 5:
            carriersDead += 1

    return patrolsDead, submarinesDead, destroyersDead, battleshipsDead, carriersDead
```

#This function has 2 ways of using, it can check if a game is over and print tables into the terminal and the oF, it first equalizes the number of destryoed
#ships with the returned values of the function gameoverChecker() then sees if it was called to check or print, with taking care of spaces and tabs...

```
def tablePrinter(characterList,gridList,table, table2, caller):
    p1Win = False
    p2Win = False
    gameOver = False
    pat2, sub2, dest2, batt2, car2 = gameoverChecker(p2Ships, p2Table, characterList, gridList)
    pat1, sub1, dest1, batt1, car1 = gameoverChecker(p1Ships, p1Table, characterList, gridList)

    if caller == "checker":
        if pat1 == 4 and sub1 == 1 and dest1 == 1 and batt1 == 2 and car1 == 1:
            gameOver = True
            p2Win = True
            if pat2 == 4 and sub2 == 1 and dest2 == 1 and batt2 == 2 and car2 == 1:
                p1Win = True
        elif pat2 == 4 and sub2 == 1 and dest2 == 1 and batt2 == 2 and car2 == 1:
            gameOver = True
            p1Win = True
        return gameOver, p1Win, p2Win

    if caller == "printer":
        oF.write("%s\t\t%s"%("Player1's Hidden Board", "Player2's Hidden Board\n  "))
        print("%s\t\t%s"%("Player1's Hidden Board", "Player2's Hidden Board\n  "), end = "")
    elif caller == "printer2":
        oF.write("%s\t\t\t\t%s"%("Player1's Board", "Player2's Board\n  "))
        print("%s\t\t\t%s"%("Player1's Board", "Player2's Board\n  "), end = "")

    for i in characterList[:characterList.index(gridList[-1][0]) + 1]:
        oF.write("%-2s"%i)
        print("%-2s"%i, end = "")
    oF.write("\t\t  ")
    print("\t\t  ", end = "")

    for i in characterList[:characterList.index(gridList[-1][0]) + 1]:
        if i != "J":
            oF.write("%-2s"%i)
            print("%-2s"%i, end = "")
        else:
            oF.write(("%-2s"%i)[:-1])
            print(("%-2s"%i)[:-1], end = "")
    print()
    oF.write("\n")

    for i in range(1, int(gridList[-1][1:]) + 1):
        oF.write("%-2d"%i)
        print("%-2d"%i, end = "")

        for j in characterList[:characterList.index(gridList[-1][0]) + 1]:
            piece = j + str(i)
            oF.write("%-2s"%table[piece])
            print("%-2s"%table[piece], end = "")
        oF.write("\t\t%-2d"%i)
        print("\t\t%-2d"%i, end = "")

        for j in characterList[:characterList.index(gridList[-1][0]) + 1]:
            piece = j + str(i)
            if j != "J":
                oF.write("%-2s"%table2[piece])
                print("%-2s"%table2[piece], end = "")
            else:
                oF.write(("%-2s"%table2[piece])[:-1])
                print(("%-2s"%table2[piece])[:-1], end = "")
        oF.write("\n")
        print()

    print("\nCarrier\t\t%s\t\tCarrier\t\t%s"%(("X "*car1 + "- "*(1-car1))[:-1], ("X "*car2 + "- "*(1-car2))[:-
1]), end = "")
    print("\nBattleship\t%s\t\tBattleship\t%s"%(("X "*batt1 + "- "*(2-batt1))[:-1], ("X "*batt2 + "- "*(2-
batt2))[:-1]), end = "")
    print("\nDestroyer\t%s\t\tDestroyer\t%s"%(("X "*dest1 + "- "*(1-dest1))[:-1], ("X "*dest2 + "- "*(1-
dest2))[:-1]), end = "")
    print("\nSubmarine\t%s\t\tSubmarine\t%s"%((("X "*sub1 + "- "*(1-sub1)))[:-1], ("X "*sub2 + "- "*(1-
sub2))[:-1]), end = "")
    print("\nPatrol Boat\t%s\t\tPatrol Boat\t%s"%(("X "*pat1 + "- "*(4-pat1))[:-1], ("X "*pat2 + "- "*(4-
pat2))[:-1]))
    oF.write("\nCarrier\t\t%s\t\t\t\tCarrier\t\t%s"%((("X "*car1 + "- "*(1-car1))[:-1], ("X "*car2 + "- "*(1-
car2))[:-1])))
    oF.write("\nBattleship\t%s\t\t\t\tBattleship\t%s"%(("X "*batt1 + "- "*(2-batt1))[:-1], ("X "*batt2 + "-
"*(2-batt2))[:-1]))
    oF.write("\nDestroyer\t%s\t\t\t\tDestroyer\t%s"%(("X "*dest1 + "- "*(1-dest1))[:-1], ("X "*dest2 + "-
"*(1-dest2))[:-1]))
    oF.write("\nSubmarine\t%s\t\t\t\tSubmarine\t%s"%(("X "*sub1 + "- "*(1-sub1))[:-1], ("X "*sub2 + "-
"*(1-sub2))[:-1]))
```

```
   oF.write("\nPatrol Boat\t%s\t\t\tPatrol Boat\t%s\n"%(("X "*pat1 + "- "*(4-pat1))[:-1], ("X "*pat2 + "-
"*(4-pat2))[:-1]))
   if caller != "printer2":
      oF.write("\n")


#Then comes the function the checks if there are any errors in players' outputs are available and returns
the appropriate message of it
#for the user and if an error is available or not with the keyname of the move that was played.
def errorCheck(moveIndex, characterList, gridList, guessList):
   if moveIndex == len(guessList):
      return "needGuess", "NG", False
   try:
      sit = 0
      errorAv = False
      errorString = "\n"
      move = guessList[moveIndex].split(",")
      y = move[0]
      x = move[1]
      if x == "" or y == "":
         sit = 1
         raise IndexError
      str(int(y))
      keyName = x + y
      if len(move) > 2 or x not in characterList:
         raise ValueError
      assert keyName in gridList
   except ValueError:
      errorString = "ValueError: Couldnt process the value entered into the grid"
      errorAv = True

   except IndexError:
      if sit == 0:
         errorString = "IndexError: Entered less values than needed"
      elif sit == 1:
         errorString = "IndexError: Entered an empty value"
      errorAv = True
   except AssertionError:
      errorString = "AssertionError: Invalid Operation."
      errorAv = True
   except:
      errorString = "kaBOOM: run for your life!"
      errorAv = True
   else:
      return errorString, keyName, errorAv
   return errorString, "", errorAv
```

```
round = 0
moveIndex1 = 0
moveIndex2 = 0
p1Win = False
p2Win = False
gameOver = False
p1Error = False
p2Error = False

oF.write("Battle of Ships Game\n")
print("Battle of Ships Game")

#This is the main block of the program, it keeps repeating as long as the game isnt over, prints tables
based on move index for each player, prints an error
#message if there is any, and modifies both the original and visible dicts of players
while gameOver == False:
    round += 1
    if moveIndex1 == len(p1Guess) or moveIndex2 == len(p2Guess):
        break

    errorString1, keyName1, errorAv1 = errorCheck(moveIndex1, characterList, gridList, p1Guess)
    print("\nPlayer1's Move\n\nRound : %d\t\t\t\tGrid Size: %dx%d\n\n"%(round,
characterList.index(gridList[-1][0]) + 1, int(gridList[-1][1:])), end = "")
    oF.write("\nPlayer1's Move\n\nRound : %d\t\t\t\tGrid Size: %dx%d\n\n"%(round,
characterList.index(gridList[-1][0]) + 1, int(gridList[-1][1:])))
    tablePrinter(characterList, gridList, p1Table, p2Table, "printer")
    oF.write("Enter your move: %s\n"%p1Guess[moveIndex1])
    print("\nEnter your move: %s"%p1Guess[moveIndex1])

    while errorAv1 == True:
        oF.write(errorString1)
        print(errorString1, end = "")
        moveIndex1 += 1
        if moveIndex1 < len(p1Guess):
            moveIn = p1Guess[moveIndex1]
        else:
            moveIn = ""
        oF.write("\nEnter your move: %s\n"%moveIn)
        print("\nEnter your move: %s\n"%moveIn)
        errorString1, keyName1, errorAv1 = errorCheck(moveIndex1, characterList, gridList, p1Guess)


    if keyName1 == "NG":
        break

    if p2Dict[keyName1] != "-":
        p2Table[keyName1] = "X"
        p2Dict[keyName1] = "X"
    else:
        p2Table[keyName1] = "O"
        p2Dict[keyName1] = "O"
    moveIndex1 += 1

    errorString2, keyName2, errorAv2 = errorCheck(moveIndex2, characterList, gridList, p2Guess)
    print("\nPlayer2's Move\n\nRound : %d\t\t\t\tGrid Size: %dx%d\n\n"%(round,
characterList.index(gridList[-1][0]) + 1, int(gridList[-1][1:])), end = "")
    oF.write("\nPlayer2's Move\n\nRound : %d\t\t\t\tGrid Size: %dx%d\n\n"%(round,
characterList.index(gridList[-1][0]) + 1, int(gridList[-1][1:])))
    tablePrinter(characterList, gridList, p1Table, p2Table, "printer")
    oF.write("Enter your move: %s\n"%p2Guess[moveIndex2])
    print("\nEnter your move: %s"%p2Guess[moveIndex2])

    while errorAv2 == True:
        oF.write(errorString2)
        print(errorString2, end = "")
        moveIndex2 += 1
        if moveIndex2 <= len(p1Guess) - 1:
            moveIn = p2Guess[moveIndex2]
        else:
            moveIn = ""
        oF.write("\nEnter your move: %s\n"%moveIn)
        print("\nEnter your move: %s"%moveIn)
        errorString2, keyName2, errorAv2 = errorCheck(moveIndex2, characterList, gridList, p2Guess)

    if keyName2 == "NG":
        break

    if p1Dict[keyName2] != "-":
        p1Table[keyName2] = "X"
        p1Dict[keyName2] = "X"
    else:
        p1Table[keyName2] = "O"
        p1Dict[keyName2] = "O"
    moveIndex2 += 1

    gameOver, p1Win, p2Win = tablePrinter(characterList, gridList, p1Table, p2Table, "checker")


#Here comes winning conditions, if the game was won by one or both of the players it prints the
following strings with the table of final information at the end
if gameOver == True and p1Win == True and p2Win == False:
    oF.write("\nPlayer1 Wins!\n\n")
    print("\nPlayer1 Wins!\n")
    p1Table = p1Dict
elif gameOver == True and p2Win == True and p1Win == False:
    oF.write("\nPlayer2 Wins!\n\n")
    print("\nPlayer2 Wins!\n")
    p2Table = p2Dict
elif gameOver == True and p2Win == True and p1Win == True:
    print('\nIt is a Draw!\n')
    oF.write("\nIt is a Draw!\n\n")
if gameOver == True:
    oF.write("Final Information\n\n")
    print("Final Information\n")
    tablePrinter(characterList,gridList,p1Table, p2Table, "printer2")

#Name: Adam Sattout
#ID: b2220765061

oF.close()
```

## 3.1 Time Spent On This Assignment

| | |
|---|---|
| Time spent for analysis | This section includes reading the assignment pdf thoroughly, understanding how the game works, and identifying the main problem with the skill tested in this assignment. Overall I would say I have spent 4 hours in this section. |
| Time spent for design and implementation | This part might have been the longest. It includes finding a solution to the main problem with setting an algorithm to how the game would be processed. Whether it is how to take ship placements from users and defining ships, checking for players moves and if they had any problems or not, or checking for all the minor details in information presentation. This part took me nearly 12 hours. |
| Time spent for testing and reporting | The final part of this assignment was a crucial part of this project to be able to send it flawlessly. Also the reporting to inform the user and other developers on how the program would work, so I put a lot of effort in the section as well adding up to nearly 5 hours. |

For reusability, this code may be used to create any kind of system that has registering and removing clients involved. All the function utilities have been discussed in the 4.1 section.

## 4 User Catalogue

To be able to use this program, you must insert your inputs into the associated text file and run it with python3. Otherwise you may face difficulties running your program. Also it is advised to check for the game rules before playing even if the program will inform you if you had a false input every time you do something wrong to have a fluid gameplay experience. As for the input syntax, you will have to mimic the syntax I am about to provide.

### 4.1 Input Syntax

To place your ships you have to type in 10 lines in this form:

;;;;;;;;; with your desired place of a piece of a ship between the colunms

To guess a move, you have to insert a move in the form of:

[Number],[CapsLetter] with it being logical to fit in a grid of 10x10 (10,J being the furthest bottom right point and 1,A being the nearest upper left one)

To run the program, it has to be run from the terminal in the form of:

```
Python3 Assignment4.py A B C D
```

With:

A = Player1's placements file

B = Player2's placements file

C = Player1's guesses

D = player2's guesses

### 4.2 Output

For the output, you will be able to find it in the same folder of the program called "Battleship.out"

## 5 Conclusion

In this section, I would like to go through a short summary of this whole assignment and give my thoughts about it.

At the start, I gave a brief summary about what I thought about the game and the main mechanics that have to be added.

Then in section 2, I gave a detailed explanation upon the used functions in the program as well as a simple pseudocode to furthermore illustrate how they would work.

In section 3, the original code was given as well as how the work was done from my prospective as a programmer and how other programmers might use the code for a different usage.

And lastly I gave the instructions for the users that will use this program so it can be used in a correct way

| Evaluate | Points | Evaluate Yourself |
|---|---|---|
| Readable Code and Meaningful Naming | 5 | 4 |
| Using Explanatory Comments | 5 | 5 |
| Efficiency | 5 | 4 |
| Function Usage | 15 | 15 |
| Correctness, File I/O | 30 | 30 |
| Exceptions | 20 | 20 |
| Report | 20 | 20 |
| Total | 100 | 98 |