Project 1

Battleship

CIS-17C 48785

Name: Yiu,Freeman

Date: 11/5/2023

Introduction

Title: Battleship

Purpose: Sink all enemy ships to win given each party has 4 ships with different size

4 main ships:

- Carrier — 5 health points
- Battleship — 4 health points
- Submarine — 3 health points
- Destroyer — 2 health points

1. When a new game starts, it prints out the main menu which documents the rules of the game.
2. Both party's ships are randomly placed on the grid, with no collision happening
3. The player and the enemy(Computer) will take turn choosing points until one party loses the game
4. At the end of the game, the user is asked whether they want to look at the logs of the game(every moves that the player and the enemy performed)

***Recommend to enlarge the output screen as big as possible, results will show on top while the updated grid will show below of it

Summary

Total lines: approximately 800 lines
Number of classes: 3 main classes
Number of methods(including class methods): 34 functions total

This project focuses on C++ STL library, mainly the ones that are related to data structures such as maps and queues.

This project took me around 2-3 weeks to complete. During the process, most of my time was spent on organizing the classes and selecting the correct member functions. The hardest part was actually implementing the algorithm to avoid collisions and it took me numerous tries to create one. Next hardest thing was creating the player function as I have to take account of input errors. As a result, that member function is 100+ lines long and it is the longest function in the whole program.

Overall, I am slightly disappointed since it didn't turn out what I expected. However, this was a good experience as I got the opportunity to implement things that I learned before. Although I didn't use the STL library sufficiently, I transformed simple events into something that utilizes it, such as turning the horizontal titles into a list.

Description

3 main classes:

- Game: Controls formation and output of the grid
- Player: Class for the user, focuses on whether the player has hit the enemy or not
- PC: Class for the enemy, focuses on whether the enemy has hit the player or not

Mainly used maps to store the ships and their respective health points, which is very important

Generalized PseudoCode

Start Program

Initialize classes Game,Player and PC
Prompts a main menu for the user to read the rules

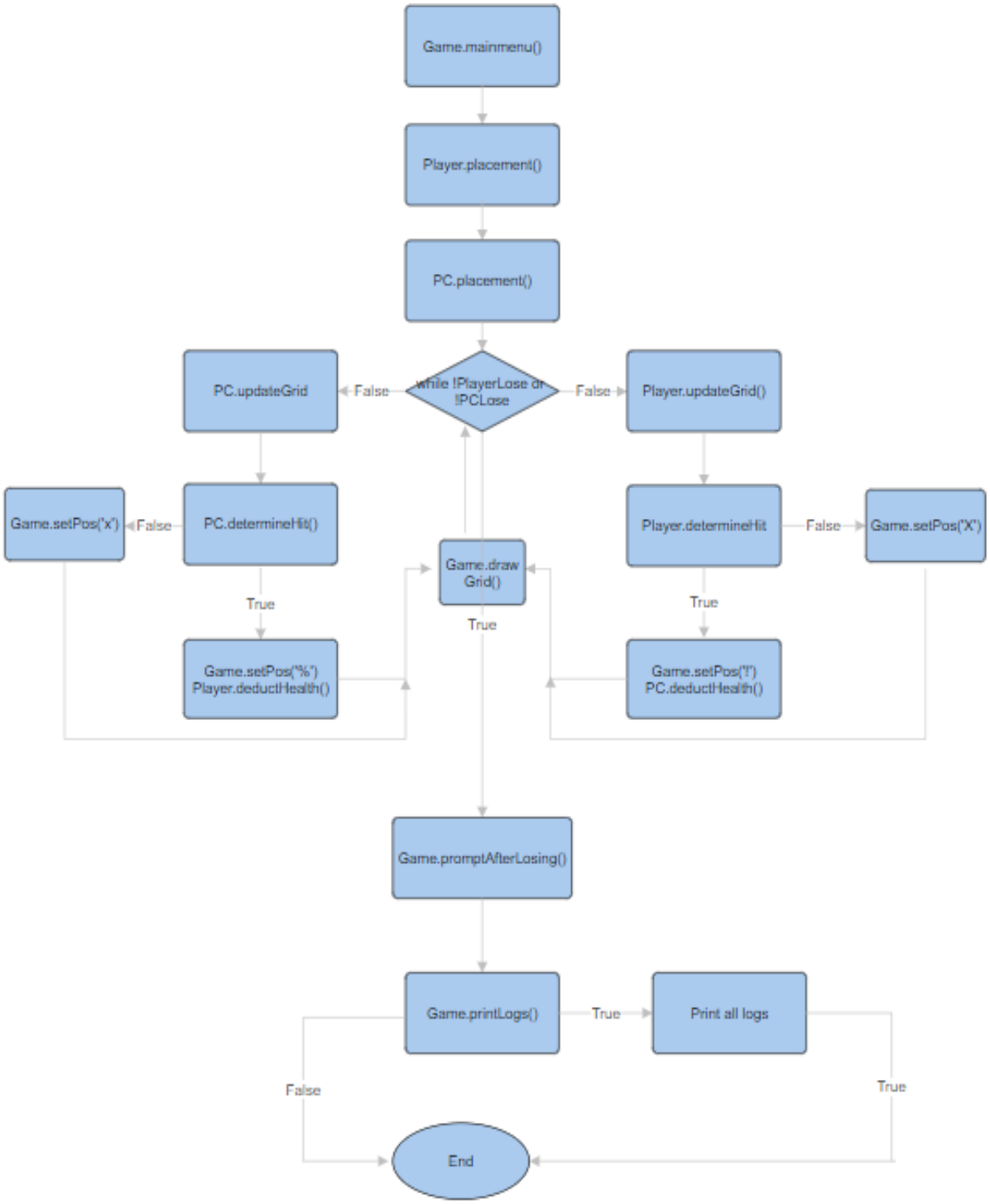
Player's ships random placement onto grid
PC's ships random placement onto grid

While Player hasn't lose or PC hasn't lose
    Player inputs position and read
        If Player hits a point
            Set that point to '!'
            Deducts a point from PC's respective ship
            Adds a sentence to logs stating Player hits at point X
        Else
            Set that point to 'X'
            Adds a sentence to logs stating the Player misses
    PC computes a random position
        If PC hits a point
            Set that point to '%'
            Deduct a point from Player's respective ship
            Adds a sentence to logs stating PC hits at point X
        Else
            Set that point to 'x'
            Adds a sentence to logs stating PC misses

Prompts the user whether they want to view the logs of the game
If yes
    Print every moves

End Program

Generalized Flowchart



Sample Input/Ourput

```
    1  2  3  4  5  6  7  8  9  10
A   C  C  C  C  C  .  .  .  .  .
B   .  .  .  .  .  .  .  .  .  .
C   .  .  .  .  .  .  .  .  .  .
D   .  .  .  .  .  .  .  .  .  .
E   .  B  B  B  B  .  .  .  .  .
F   .  D  D  .  .  .  .  .  .  .
G   .  .  .  .  .  .  .  .  .  .
H   .  .  .  .  .  .  .  .  .  .
I   .  .  .  S  S  S  .  .  .  .
J   .  .  .  .  .  .  .  .  .  .

Please enter a position: B 1
```

Input: B 1

Output

```
     *****Results*****
--------------------------

--Player's side:

You missed!


--Enemy's side:

The enemy chooses point G10
The enemy misses



    1  2  3  4  5  6  7  8  9  10
A   C  C  C  C  C  .  .  .  .  .
B   X  .  .  .  .  .  .  .  .  .
C   .  .  .  .  .  .  .  .  .  .
D   .  .  .  .  .  .  .  .  .  .
E   .  B  B  B  B  .  .  .  .  .
F   .  D  D  .  .  .  .  .  .  .
G   .  .  .  .  .  .  .  .  .  x
H   .  .  .  .  .  .  .  .  .  .
I   .  .  .  S  S  S  .  .  .  .
J   .  .  .  .  .  .  .  .  .  .

Please enter a position:
```

Checkoff List

- List
    - Used to store the title of the horizontal column(numbers)
    - Located at Game.drawGrid() and the constructor of class Game
- Set
    - Used it to store forbidden characters of the two parties
    - Located at PC.updateGridE() and Player.updateGrid()
- Map
    - Purpose is to store key-value pair, ship character being the key and health being its value
    - Located at the constructors of classes Player and PC
- Stack
    - Push an integer whenever the player and PC makes a move, use it to return the total number of moves of the game
    - Located at PC.updateGridE() and Player.updateGrid(), specifically after the Player/PC chooses a valid position
- Queue
    - Store the message of every moves performed in the Game, use it to print out all the moves if the user requested to do so
    - Located at Game.printLogs()

- Trivial Iterator
    - An iterator used to traverse a container. For example, traversing over a map to look for its values based on the key.
    - Located at Player.placement or PC.placement where it iterates over the map of ships, for(auto i=ships.begin();i!=ships.end;i++).
- Input Iterator
    - An iterator that reads the value only once and then incremented. Weakest iterator out of all types of iterators, related to istream.
    - The increment inside a for-loop is an example of input iterator
    - Located at any for-loops in the program. For example: for(int i=0;i<ROWS;i++)
- Output Iterator
    - Opposite of Input Iterators and can modify values of the container, but it is still a one-way iterator, related to ostream.
    - Inserting elements into a stack is an example of output iterator as we are modifying the container, which is the purpose of output iterators

    - Located at player.updateGrid() or PC.updateGrid() where it states game.pushStack() whenever a move has been made
- Bidirectional Iterator
    - An iterator that can move back and forth
    - An example of a bidirectional container is accessing a list, it can iterate from the beginning to the end and from the end to the beginning.
    - Located at Game.drawGrid where for(auto i: horzRow){...}, horzRow is a list
- Random Access Iterator
    - An iterator that accesses items at a random position. Strongest iterator out of all types
    - STL random_shuffle function is an example of random access Iterator
    - Located at PC::uupdateGridE() definition where random_shuffle(copyletters,copyletters+10)
- Count
    - Purpose is to see if the user's/PC's given position is a target character, which occurs at least once
    - Located at edetermineHit() and pdetermineHit(), if(count(hitchars,hitchars+4,c)>0){return true;}, c is the position character and hitchars is an array of target characters
- For_each
    - Use it to print the barrier(formatting the game), which is a string made up of '.' characters
    - Located at Player.updateGrid(), where it prints the barrier when the player misses, for_each(spacing,spacing+27,printBarrier)
- Find
    - Purpose is to find if the user's/PC's given position is a forbidden position(contains a forbidden character), function returns true if it contains one. Otherwise, it returns false.
    - Located at PC.updateGridE() and Player.updateGrid(), specifically where the function detects the error
    - Function calls to pdetermineError() and edetermineError() and return true if the position is forbidden, which is if(s.find(c)!=end){return true;}
- Copy

    - Used to store the letters of the vertical column(A-J)the PC can choose
    - Copy the original and will random-shuffle it later to make things randomized
    - Located at PC.updateGridE()
- Fill
    - Purpose is to fill an array of '-' characters to format the results, acts as a line break to make things tidier
    - Located at Player.updateGrid() as the player goes first and modifies the output to make it look less messy
- Random_Shuffle
    - With a copy of the letter array for the PC to choose, it will random shuffle the array and a random element is computed
    - Located at PC.updateGrid, specifically when the PC randomly chooses its horizontal-row position(a letter)
- Sort
    - Initially inserts titles 1-10 into the vertical column list out of order;
    - Sorting is used to sort the list and compute the titles respective to their order
    - Located at the constructor of Game