Project 2

Battleship V2

CIS-17C 48785

Name: Yiu,Freeman

Date: 12/11/2023

Introduction

Title: Battleship V2

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 1050 lines
Number of classes: 5 main classes
Number of methods(including class methods): 54 functions total

This project focuses on key data structures, such as trees and recursion. C++ STL library functions are also used, including maps and stacks.

The project took me around 1 week to implement new data structures into the project. While developing, I spent most of my time on parts related to trees and graphs. Since graphs were very new to me, I had to research a lot of information about them. Besides adding the new topics, I also fixed minor bugs from Battleship V1 and improved the interface, such as making the grid more tidy and displays the number ships remaining/

Overall, I am still slightly disappointed since it didn't turn out what I expected. However, I was able to add new topics that I have learned into my previous project. This process forced me to think critically on how these new data structures relate to Battleship V1. Overall, this was a very good experience and I hope the experiences gained in this project can make me a better programmer in the future.

Description

5 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.
- Node: A structure of data, used for Tree class for implementing a tree
- Tree: A collection of nodes, used to search a value in O(log(N)) time
- Graph: A collection of vertices, used to find the weight of a pair

Generalized PseudoCode

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

For each Player's ship stored in a map
    Assign a random position to the grid with respective ship character
    Choose a random number between 1 and 2
    If the number is 1(Vertical)
        Compute a random col(y-axis) between 0 to 10-shipSize
        Compute a random row(x-axis) between 0-10
        While collision occurs in vertical position
            Re-Compute another number for row and column
        Loop and assign the ship characters into grid
    Else
        Compute a random row(x-axis) between 0 to 10-shipSize
        Compute a random col(y-axis) between 0-10
        While collision occurs in horizontal position
            Re-Compute another number for row and column
        Loop and assign the ship characters into grid

For each PC's ship stored in a map
    Assign a random position to the grid with respective ship character
    Choose a random number between 1 and 2
    If the number is 1(Vertical)
        Compute a random col(y-axis) between 0 to 10-shipSize
        Compute a random row(x-axis) between 0-10
        While collision occurs in vertical position
            Re-Compute another number for row and column
        Loop and assign the ship characters into grid

Else

Compute a random row(x-axis) between 0 to 10-shipSize
Compute a random col(y-axis) between 0-10
While collision occurs in horizontal position
Re-Compute another number for row and column
Loop and assign the ship characters into grid

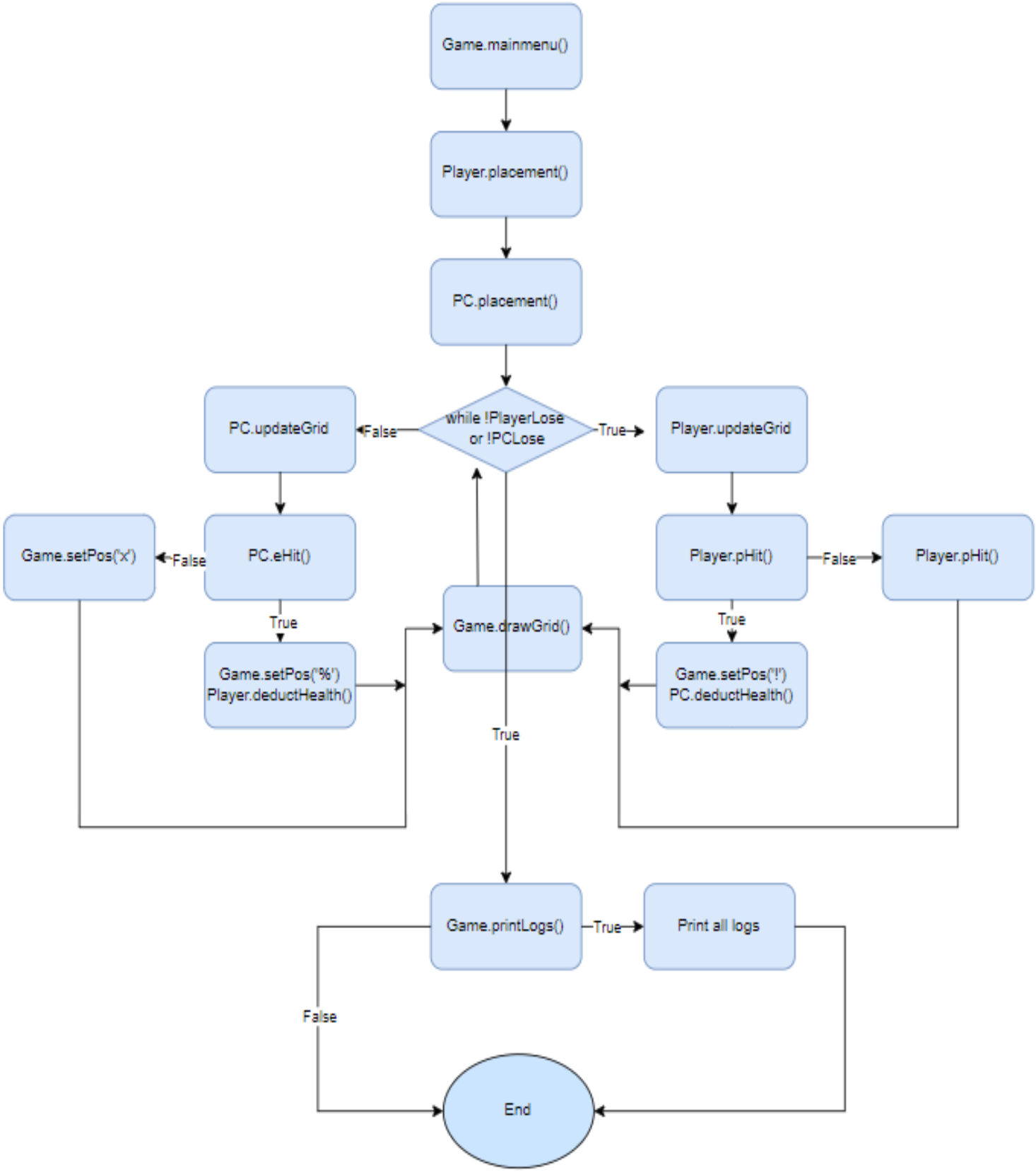
While Player hasn't lose or PC hasn't lose
Player inputs position and read
While player's position is out of bounds
Prompts to enter a new position
If Player hits a point
Set that point to '!'
Deducts a point from PC's respective ship
Adds a sentence to logs stating Player hits at point X
Else
Set that point to 'X'
Adds a sentence to logs stating the Player misses

PC computes a random position
While computed position is out of bounds
Re-compute a new 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
Prints updated Grid

Prompts the user whether they want to view the logs of the game
If yes
Print every moves
Prompts the user if they want to look at a specific move
If yes
Print the specific move

End Program

Generalized Flowchart



Sample Input/Output

```
  |   1  2  3  4  5  6  7  8  9  10
-----------------------------------
A |   .  .  .  C  C  C  C  C  .  .
B |   .  .  .  .  .  .  .  .  .  .
C |   .  .  .  .  .  .  .  .  .  .
D |   .  .  .  .  .  .  .  .  .  .
E |   .  .  .  .  .  .  .  .  .  .
F |   .  .  .  .  .  .  .  .  .  .
G |   .  S  S  S  .  .  .  .  .  .
H |   .  .  B  B  B  B  .  .  .  .
I |   D  D  .  .  .  .  .  .  .  .        Number of enemy ships remaining: 4
J |   .  .  .  b  b  b  b  .  .  .        Number of player ships remaining: 4


Please enter a position:
```

Input: B 1

Output

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

--Player's side:

You missed!


--Enemy's side:

The enemy chooses point C6
The enemy misses



  |   1  2  3  4  5  6  7  8  9  10
-----------------------------------
A |   .  .  .  C  C  C  C  C  .  .
B |   X  .  .  .  .  .  .  .  .  .
C |   .  .  .  .  .  x  .  .  .  .
D |   .  .  .  .  .  .  .  .  .  .
E |   .  .  .  .  .  .  .  .  .  .
F |   .  .  .  .  .  .  .  .  .  .
G |   .  S  S  S  .  .  .  .  .  .
H |   .  .  B  B  B  B  .  .  .  .
I |   D  D  .  .  .  .  .  .  .  .        Number of enemy ships remaining: 4
J |   .  .  .  b  b  b  b  .  .  .        Number of player ships remaining: 4


Please enter a position:
```

Checkoff List

- 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
    - map<char,int> ships
- 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
    - stack<int> numMoves
- 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()
    - queue<string> log

- 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::updateGridE() definition where random_shuffle(copyletters,copyletters+10)

C++ STL

- 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(), copy(letterchoices,letterchoices+10,copyletters)
- 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), random_shuffle(copyletters,copyletters+10)
- Sort
    - Initially inserts titles (A-J) 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 Game.drawGrid() as it prints out the column, sort(verticalChar,verticalChar+10), vertChar is an array of characters from A to J.

*****Requirement (Recursion, Trees, etc,)**

Recursive Sort

- MergeSort
  - Sorts the horizontal labels and vertical labels of the grid
  - Uses templates as the horizontal labels are integers and vertical labels are chars. Templates can handle different data types
  - Function call:
    - mergesort<int>(horzRow,0,ROWS);
    - mergesort<char>(vertCol,0,COLS);
  - Definition:
    - template <class T>
      void mergesort(T *arr,int l,int r){
      if(l<r){
      int mid=l+(r-l)/2;
      mergesort(arr,l,mid);
      mergesort(arr,mid+1,r);
      merge(arr,l,mid,r);
      }
      }
    - template <class T>
      void merge(T *arr,int l,int m,int r){
      int low=m-l+1;
      int high=r-m;
      T arrl[low],arrR[high];
      for(int i=0;i<low;i++){
      arrl[i]=arr[l+i];
      }
      for(int j=0;j<high;j++){
      arrR[j]=arr[m+1+j];
      }
      int i=0,j=0,k=l;
      … (For full definition, refer to “Functions.h” for more info)

Recursion

- Print Barrier
    - Printing the barrier for the results to make the portion more tidy. Uses a recursive approach instead of using loops
    - Function call
        - printBarrier('-',NUMBARRIER);
        - NUMBARRIER is the number of times to print '-' recursively. In this case, it is 27
    - Definition
        - void printBarrier(char c,const int num){
          if(num<=0)return;
          printBarrier(c,num-1);
          cout<<c;
          }

Tree

- Binary Tree and Binary Search Tree
    - Adds forbidden characters of both classes into a tree to determine if the given point is forbidden or not. When a player/enemy enters a position, it searches if the forbidden character exists in the tree
    - Calls
        - fbc=new Tree(); //Player's forbidden tree
          fbc->root=fbc->insert(fbc->root,'C');
          fbc->root=fbc->insert(fbc->root,'B');
          fbc->root=fbc->insert(fbc->root,'D');
          …
        - efbc=new Tree(); //Enemy's forbidden tree
          efbc->root=efbc->insert(efbc->root,'c');
          efbc->root=efbc->insert(efbc->root,'b');
          efbc->root=efbc->insert(efbc->root,'d');
          …
    - Definition(insert and find):
        - Node* Tree::insert(Node* root,char c){
          if(!root){

```
        root=new Node(c);
        return root;
    }
    else if(c < root->val){
        root->left=insert(root->left,c);
        root=balance(root);
    }
    else if(c > root->val){
        root->right=insert(root->right,c);
        root=balance(root);
    }
    return root;
}
```

- bool Tree::find(Node* root,char c){
  ```
    if(!root){
        return false;
    }
    if(root->val==c){
        return true;
    }
    if(c<root->val){
        return find(root->left,c);
    }
    else {
        return find(root->right,c);
    }
  }
  ```

Graphs

- Graph with vertices and weights
  - Return a number based on its vertices. The graph has vertices {‘A’,’A’} to {‘A’,’J’}, each pair has a unique weight that represents its return values, acts like a character to integer converter but uses graphs.

- Call
  - vert=toupper(vert);
    ver=g.findweight({'A',vert}); //Returns the weight of that vertices

- Definition
  - Graph(int num){ //Located in ‘Game.h” file
    ```
    v=num;
    for(int i=0;i<v;i++){
        ctoi[{'A','A'+i}]=i;
    }
};
```

  - int Graph::findweight(pair<char,char> val){
    ```
    if(ctoi.find(val)==ctoi.end())return -1;
    return ctoi[val];
    }
    ```

    //ctoi is a map with the vertices as keys and weights as values