A New Battleship Game

A NEW BATTLESHIP GAME

A new version of the battleship game is proposed.

As in the original game, each player has a rectangular board where he/she positions his/her ships and the objective of the game is to destroy the opponent's ships.

The differences from the original battleship game are the following:

- in each turn, the player can see the opponent's board;
- an attack to the opponent is made by sending a bomb, aiming at a specified position of the opponent's board; however, due to manufacturing problems the bomb may not fall exactly at the specified position;
- during the trajectory of the bomb the opponent's ships may move;
- each player plays just once, before passing the turn to the opponent;
- a ship is considered to be destroyed when at least half of the ship is destroyed.

Each player's fleet has several types of ships, as defined in the first practical work.

Both players have the same number and type of ships. Before the game begins, each player chooses the name of the file that contains his/her board, prepared using the program developed in the first practical work. That program must be slighty modified, as described later.

The game ends whenever one of the fleets is destroyed. The criterion for deciding that a fleet is destroyed is also different from the original game, as explained later.

DEVELOPMENT OF THE PROGRAM

General operation

The program must start by reading the name of each player and the name of the board file that he/she wants to use. After choosing, randomly, the starting player, it must iteratively do the following, until one of the fleets is destroyed.

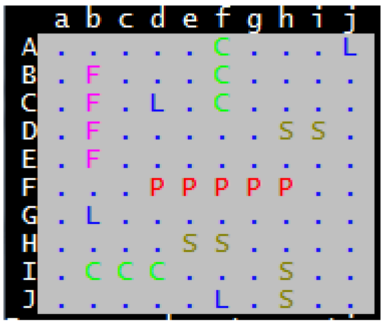
- show the opponent’s board;
- ask the current player the coordinates of the target that he/she wants to attack and create a bomb to attack the chosen target;
- attack the opponent’s board, using the created bomb.

A fleet is destroyed when all the ships are destroyed; a ship is destroyed when at least half of the ship is destroyed.

The program must also keep a log of the 10 best scores obtained by the winners of the game. The score of a winner is determined using the following formula: T*S/A, where T is the time taken to destroy the opponent’s fleet, S is the total area occupied by the ships and A is the area of the board. The lower the score the best is the result. The log, ordered from best to worst scores, must contain the following data: the name of the player, the obtained score, the dimensions of the board and the total area occupied by the ships.

Board file

The board files must have the format specified in figure 1.b. The format is similar to the one that was specified for the first practical work, with an additional information for each ship (the last value in each row) that specifies the color code of the ship (example: the color code for light red is 12, …). A slight modification the previously developed program is necessary to create these new board files.



```
  a b c d e f g h i j
A . . . . . C . . . L
B . F . . . C . . . .
C . F . L . C . . . .
D . F . . . . . S S .
E . F . . . . . . . .
F . . . P P P P P . .
G . L . . . . . . . .
H . . . . S S . . . .
I . C C C . . . S . .
J . . . . . L . S . .
```

a) Final aspect of the board after the positioning of all ships.

```
10 x 10
P 5 Fd H 12
F 4 Bb V 13
C 3 Af V 10
C 3 Ib H 10
S 2 He H  6
S 2 Ih V  6
S 2 Dh H  6
L 1 Cd H  9
L 1 Aj V  9
L 1 Jf V  9
L 1 Gb H  9
```

b) New board file corresponding to the board of a).

Figure 1

The first row of the file contains two numbers representing, respectively, the number of rows and columns of the board (10 rows x 10 columns, in the example shown). So that the lines and columns may be indexed using letters of the alphabet (upper case for the rows and lower case for the columns), the maximum number of rows/columns is 26.

NOTE: the symbols used in the examples correspond to the Portuguese names of the vessels; they can be replaced by symbols corresponding to the English names.

Classes definitions

You can find below a first, partial definition of structs and classes that can be used in the program:

```
//==========================================================================
struct PositionChar      // to store a board position in char format
{                        // example: 'B','d'
  char lin, col;         // ranges: lin - ['A'..'Z']; col - ['a'..'z']
};
//--------------------------------------------------------------------------
struct PositionInt       // to store a board position in unsig. integer format
{                        // example: 7,0
  unsigned int lin, col; // ranges: lin - [0..26]; col - [0..26]
};
//==========================================================================
class Bomb
{
public:
  Bomb(PositionChar targetPosition);
  PositionChar getTargetPosition() const;
  // OTHER METHODS, if necessary
  // ...
  void show() const; // shows the attributes of the bomb (for debugging)
private:
  char targetLine, targetColumn;
};
//==========================================================================
class Ship
{
public:
  Ship(char symbol, PositionChar position, char orientation, unsigned int size,
unsigned int color);
  // OTHER METHODS, if necessary
  // ...
  bool move(char direction, bool rotate, unsigned int lineMin, unsigned int
columnMin, unsigned int lineMax, unsigned int columnMax); // moves the boat (SEE
NOTES)
  bool moveRand(unsigned int lineMin, unsigned int columnMin, unsigned int
lineMax, unsigned int columnMax); // moves the ship randomly
  bool attack(size_t partNumber); //partNumber = {0,1,…, size-1}
  bool isDestroyed() const;  // checks whether the ship is destroyed
  void show() const; // shows the attributes of the ship (for debugging)
private:
  char symbol; // 'P' = "porta-aviões"; 'F' = "fragata"; … (Portuguese names)
  PositionInt position; // coordinates of the upper left corner of the ship
  char orientation; // 'H' = horizontal; 'V' = vertical
  unsigned int size; // number of cells occupied by the ship, on the board
  unsigned int color; // color code: o=BLACK, 1=BLUE, … (see annex of 1st proj.)
  string status; // status[i]: upper case = good; lower case = damaged
                 // ex: "FFFF" means that the "fragata" is intact;
                 // ex: "FFfF" means that the 'partNumber' 2 was hit by a bomb
  // OTHER ATTRIBUTES OR METHODS, if necessary
  // ...
};
//==========================================================================
class Board
{
public:
  Board(const string &filename); // loads board from file 'filename'
  bool putShip(const Ship &s); // adds ship to the board, if possible
  void moveShips(); // tries to randmonly move all the ships of the fleet
  bool attack(const Bomb &b);
  void display() const; // displays the colored board during the game
  void show() const; // shows the attributes of the board (for debugging)
  // OTHER METHODS, if necessary
  // ...
private:
  int numLines, numColumns;  // redundant info …
  vector<Ship> ships;        // ships that are placed on the board
  vector<vector<int>> board; // each element indicates
                             // the index of a ship in the 'ships' vector
                             // (in the range 0..ships.size()-1) ;
                             // -1 is used to represent the sea
};
//==========================================================================
class Player
{
public:
  Player(string playerName, string boardFilename);
  void showBoard() const;   // shows the player's board
  Bomb getBomb() const;     // asks bomb target coordinates and creates the bomb
  void attackBoard(const Bomb &b); // "receives" a bomb from the opponent;
                            // updates own board taking into account the damages
                            // caused by the bomb; BEFORE THAT… moves the ships
private:
  string name; // name of the player
  Board board; // board of the player
};
```

Notes on the classes definitions

`class Bomb`

- For "manufacturing a bomb" the user must specify the position of the opponent's board that he/she wants to attack using that bomb; in order to simulate some "manufacturing problems", the coordinates (line and column of the target position) may be not those that were specified by the player. The offset between the specified and the generated coordinates may be null or, at most, one position in the North, South, East, or West direction; the offset is chosen randomly.

`class Ship`

- The **`move()`** method displaces the ship one position in the North, South, East, or West direction and/or rotates the ship, as specified by the **`direction`** and **`rotation`** parameters; the rotation center is the upper left corner of the ship, i.e, it is made by changing the orientation of the ship from '**`H`**' to '**`V`**' or from '**`V`**' to '**`H`**'.
- The **`moveRand`**() method displaces the ship one position in the North, South, East, or West direction and/or rotates the ship. The selection of the move is made in a random way. Suggestion: generate a

random number in the range 0..4, to select “no move” or the movement direction (0=“no move”, 1=‘N’, 2=‘S’, 3=’E’, 4=’W’) and another random number to select “rotation / no rotation”.

- The first four parameters of the above methods (**`minLine, minColumn, maxLine`** and **`maxColumn`**) must be used to check whether, after the movement, any of the parts(cells) of the ship would be outside the board; in this case the ship should not be moved. These methods must return a boolean value indicating whether the requested movement was possible or not.
- The **`attack()`** method must receive as parameter the **`partNumber`**, or index, of a part of the ship (a number in the range **`0..size-1`**) and, if the parameter is in that range, it must register in attribute **`status`** that the indicated part was hit. When a new ship is created the **`status`** string must be equal to **`string(symbol,size)`**, where **`symbol`** and **`size`** are the symbol and size of the ship (ex: “**`FFFF`**” for a frigate of size 4). When a part of the ship is hit the corresponding element of **`status`** must be converted to lower case (ex: when part 3 of the frigate is hit by a bomb, the **`status`** string becomes “**`FFFf`**”). The method returns a boolean value indicating whether **`partNumber`** is valid or not.
- The **`isDestroyed()`** method checks whether the ship is destroyed. A ship is considered to be destroyed if at least 50% if its parts (/cells) were hit by a bomb.

`class Board`

- Note that the attribute **`board`** is a 2D vector of int. Each cell of **`board`** must contain a number, **`n`**, in the range **`0..ships.size()-1`** if the cell is occupied by a ship having index **`n`** in the **`ships`** vector, or **`-1`**, if it corresponds to the sea.
- Vector **`ships`** should be constructed using the information contained in the board file, prepared using the program developed in the first project.
- Method **`putShip()`** must update both the **`board`** and the **`ships`** attributes of the class. If the ship does not fit inside the board or collides with other already placed ship(s) those attributes must not be updated and the function must return **`false`**, returning **`true`** otherwise.
- Method **`attack()`** must determine if there is a ship at the target coordinates of the bomb, and attack it (calling method **`attack()`** of the **`Ship`** class). It must return **`true/false`** depending on whether the target coordinates are inside the board or not.

`class Player`

- Method **`getBomb()`** must interact with the player, asking him/her which are the coordinates of the target position, in the opponent’s board, that he/she wants to hit. Then it must crate and return a bomb, aiming at the specified coordinates. Note that, as said before, the bomb may have not the desired quality …
- Method **`attackBoard()`** "receives" a bomb that was sent by the opponent player and updates the own **`board`** and **`ships`**, taking into account the damages caused by the bomb, but, before accounting for the damages, it moves his/her own ships, in a random way, trying to escape from the bomb.
- It may be necessary to use an auxiliary board to make a preview of the displacement of the ships.

Additional data structures may be necessary to accomplish the project specifications, namely the log of the 10 best scores.

Other notes

- Suggestion: implement the classes in the above specified sequence and thoroughly test all the methods before proceeding.
- You are free to add other attributes and other public or private methods to the specified classes or to modify the parameters (number and type) or the return type of the class methods.
- The code must be compilable separately (write a **`.h`** and a **`.cpp`** file for each class).
- After templates are introduced in the lectures, create a template struct for representing a **`Position`** and modify the rest of the code accordingly. Also, do the overloading of **`operator<<`** for all classes.

CODE DEVELOPMENT & GRADING

- The code must be developed both during classes and out of classes.
- The performance of each group during the classes, as well as the evolution of the code between consecutive classes, will be taken into account for the grading of this practical work.
- The elements of a group may have different gradings, depending on each one’s performance.

WORK SUBMISSION

- Create a folder with the name **`TxGyy`**, in which **`x`** represents the class number (Portuguese "turma") and **`yy`** represents the group number, for example, **`T1G05`**, for group 5 from class ("turma") 1 (always use 2 digits for the group number), and copy to that folder the source code of your program (only the files with extension **`.cpp`** and **`.h`**, if existing). Include also a file, **`ReadMe.txt`** (in simple text format), indicating the development state of the program, that is, if all the objectives were accomplished or, otherwise, which ones were not achieved, and also what improvements were made, if any. This file may contain additional information describing and justifying any decisions that you have made about the functioning of the program or its implementation (ex: modifications in the definition of the classes).
- Compress the content of the folder into the file **`TxGyy.zip`** or **`TxGyy.rar`** and upload that file in the FEUP Moodle's page of the course. Alternative ways of delivering the work will not be accepted.
- Work submission: two submissions must be done, an intermediate submission and the final one.
  - Deadline for the intermediate submission: May/11th, at 23:55;
  - Deadline for the final submission: May/25th, at 23:55.