Programming Laboratory Course
Final Project – Naval Battle

You're asked to develop a C++ program that implements an advanced version of the Battleship game. The rules have been modified, so please pay attention to the game description.

Rules of the game

Each player has two 12x12 grids. A defense grid is used to place your naval units, and an attack grid is used to track hits (X) and misses (O) on enemy units. Each player also has eight units of three different types. The three unit types are: battleship (C), support ship (S), and scout submarine (E). Below is an example of a grid:

TO		S	S	S								
B						C	C	C	C	C		
C												
D		C				S	S	S				
AND		C										
F		C						C	C	C	C	C
G		C										
H		C									AND	
THE												
THE		AND				S	S	S				
M												
N												
	1	2	3	4	5	6	7	8	9	10	11	12

Defense grid

TO												
B							OR					
C			X	X	X	X	X	OR				
D							OR					
AND	OR											
F												
G	OR		OR					OR			OR	
H												
THE			OR									
THE			OR					X	X			
M												
N												
	1	2	3	4	5	6	7	8	9	10	11	12

Attack grid

Figure 1. Example of defense grid (right) and attack grid (left)

Naval units

All naval units can perform a***action***However, this varies depending on the unit type. Furthermore, each unit has a specific size, which is also equivalent to its initial armor. When all the squares a unit occupies are hit, the unit is considered sunk. Armor and size are initially equivalent. As the unit is hit, its armor decreases while its size remains the same. The characteristics of each naval unit are listed below:

Naval unit	Amount	Size	Armor	Action
Battleship	3	5	5	Fire
Support vessel	3	3	3	Move and repair
Exploration submarine	2	1	1	Move and search

The***actions***that the different naval units can perform are reported below:

- Fire : the**battleship**Fires at specific coordinates. The shot hits if the hit square is occupied by an enemy unit. If the target unit's armor reaches 0 (all squares containing the unit have been hit), the target unit is sunk.

- Move and repair : the**support vessel**It moves to the given coordinates. Obviously, the space must not already be occupied. Once it has moved, it provides support to all the player's units present in a 3x3 area, considering the support ship itself as the center. Units in this area will be completely repaired (i.e., their armor returns to its initial value). Only one space within the area is sufficient for the entire unit to receive support. The supporting ship cannot benefit from its own repairs (so, to be repaired, it must benefit from the assistance of another support ship).

- Move and search : The**submarine**It moves to the received coordinates. Obviously, the space must not already be occupied. Once it has moved, it emits a sonar ping that reveals whether there are enemy units in a 5x5 area, centered on the submarine itself. Spaces containing enemy units (or parts of enemy units) are marked on the attack grid with the Y symbol.

Note on the movement:
Movement is relative to the unit's center square. Each movement must be valid, meaning the unit must be entirely within the grid after movement and cannot overlap with other player units. There is no rotation; units move rigidly on the grid.

Players and shifts

There are always only two players. One of the two players can be human. At the start of the game, all players place units on their defense grid. The program randomly decides which player starts the game. On their turn, the player must make a***action***to only one of its units. Once the***action***it's done, the turn passes to the other player.

Note on computer player intelligence:
We don't ask you to implement intelligent algorithms for the computer. On each turn, the computer player randomly selects one of his units and executes it.***action***.

Game controls

Each player has the***action***to one of your drives, using a command with the following syntax:

XYOrigin XYTarget

XYOrigincorrespond to the coordinates of the unit that must perform the action (the coordinates that identify a unit are those of the central box).XYTargetcorrespond to the Coordinates that are the target of the unit's action. In the case of a battleship, they correspond to the target square of fire; in the case of support ships and scout submarines, they correspond to the destination square of the movement.
Taking Figure 1 as an example, some examples of commands:

B8 G10 ->Battleship on B8 fires on square G10
L7 I2 ->Support ship in L7 moves to I2 and provides assistance (repairs) the unit in F2 and L2

With a special command (AA AA)It is possible to clear sonar spots (Character Y) in the attack grid (since units have moved, and the spots may not be updated). This command does not use the turn move.

<u>Initial positioning of naval units</u>

The initial positioning of naval units is done at the beginning of the game. In this phase of the game, the program asks the player to provide the bow and stern coordinates of each of their units. Units can only be arranged horizontally or vertically and cannot overlap (not even in the case of submarines, for simplicity's sake). For example (referring to Figure 1):

```
> > What are the coordinates for battleship 1:
> > B6 B10
> > What are the coordinates for battleship 2:
> > D2 H2
...
```

Viewing

In the case of a single-player game, you can ask the program to display the defense and attack grids using a special command:

XX XX

The display is done by printing a character for each naval unit in the defense grid and for each hit made and for the sonar readings in the attack grid, as shown in Figure 1. Each grid must have the coordinates as shown in Figure 1. The characters on the grids have the following coding:

Defense grid	
Battleship	C
Support vessel	S
Exploration submarine	AND

Attack grid	
Unit hit	X
Waterfall	OR
Sonar detection	Y

If no units are present, no hits have been made, or no detections have been made, a space is printed. Boxes corresponding to parts of units that have been hit are marked with a lowercase letter.

Matches

There are two types of matches:

- computer vs. computer games;
- player vs. computer games.

The program must handle both. For games between two computers, you must set a maximum number of turns, beyond which the game is void. During each game, a log file must be created, listing all the commands sent, in order. The program must be able to replay a game given a log file, which must be a text file with a .txt extension.

Replay

The project must include a module (consisting of a separate executable, see next point) for replaying the game, performed by reading the relevant log. The replay is achieved by printing the two game grids for each round. Printing is done on the screen, with an appropriate pause (e.g., 1 second) between each round, or to a file—in which case, writing occurs without pauses.

Executable

The project must generate two executables:

- the game executable, called "battleship", which must handle command line arguments, according to the following scheme:
  - PC topic: player vs. computer match (p stands for player, c for computer);
  - cc topic: computer vs. computer game;
- the replay executable, called "replay", which accepts command-line arguments according to the following scheme:
  - argument v [log_file_name]: prints the replay of the indicated log file to the screen;
  - argument f [log_file_name] [replay_output_file_name]: writes the replay of the indicated log file to file.

Development Notes

The project developed**must**It can be managed via CMake and compiled on the Taliercio.2020 virtual machine. You don't need to develop everything on the virtual machine, but you should periodically check to ensure you haven't used non-standard code.

Instructions for carrying out the procedure

The project must be split into multiple source files.**Each file must be written by only one student**However, you can check that your groupmates' code is working properly. An error in one file that affects the project's functionality could result in a grade penalty for the entire group.**The author's name must be**

indicated in a comment at the beginning of the file Of course, it's necessary and positive for each group to discuss how to develop the code, but each student is responsible for managing the code they write.

The following will be evaluated:

- Clarity and correctness of the code;
- Correct management of memory and data structures;
- Efficiency of the code and the solution found;
- Use of appropriate tools.

The code must be adequately commented.

The software must be based solely on the standard C++ library.

Plagiarism

The code will be checked for plagiarism between groups on the same channel and across different channels. A check will also be performed against existing code available online.

Delivery

The assignment must be submitted on Moodle (group submission as for the midterm exam), uploading an archive that includes a single directory containing:

- The source code (possibly organized into subdirectories);
- The CMakeLists.txt file needed for compilation (only one and placed in the main directory of the project);
- A log file from a computer vs. computer game and a log file from a computer vs. player game;
- A readme.txt file where you can include any notes you want to share while the code is being corrected. This file isn't documentation, which should be included as comments in the code, but rather a list of additional notes for the correctors, such as issues encountered but not resolved.

The archive must not contain the executable, because the source will be compiled during the debugging phase. The CMake system must compile with the optimization options enabled (-O2).
After uploading to Moodle,**check what you have delivered**with the following steps:

- Download your project from moodle in a different directory than the one used for development;
- Launch cmake;
- Compile the code and verify that it runs correctly.

It is recommended that students submit incomplete assignments. However, the software submitted must be compilable and executable.

Late delivery of papers will be**heavily penalized**and will only be considered within a very short time after the deadline (a few minutes). Deliveries made after that will be ignored.