How does our game work? What are its characteristics, what are its main features? How did we approach the client/server exchanges? How to understand our project or how to play, very simply? All this will be summarized in this Software Engineering technical report.

BattleShip Project

Technical Report

Thomas Pasquier and Dan Lipskier

efrei PARIS
Ecole d'ingénieurs du numérique

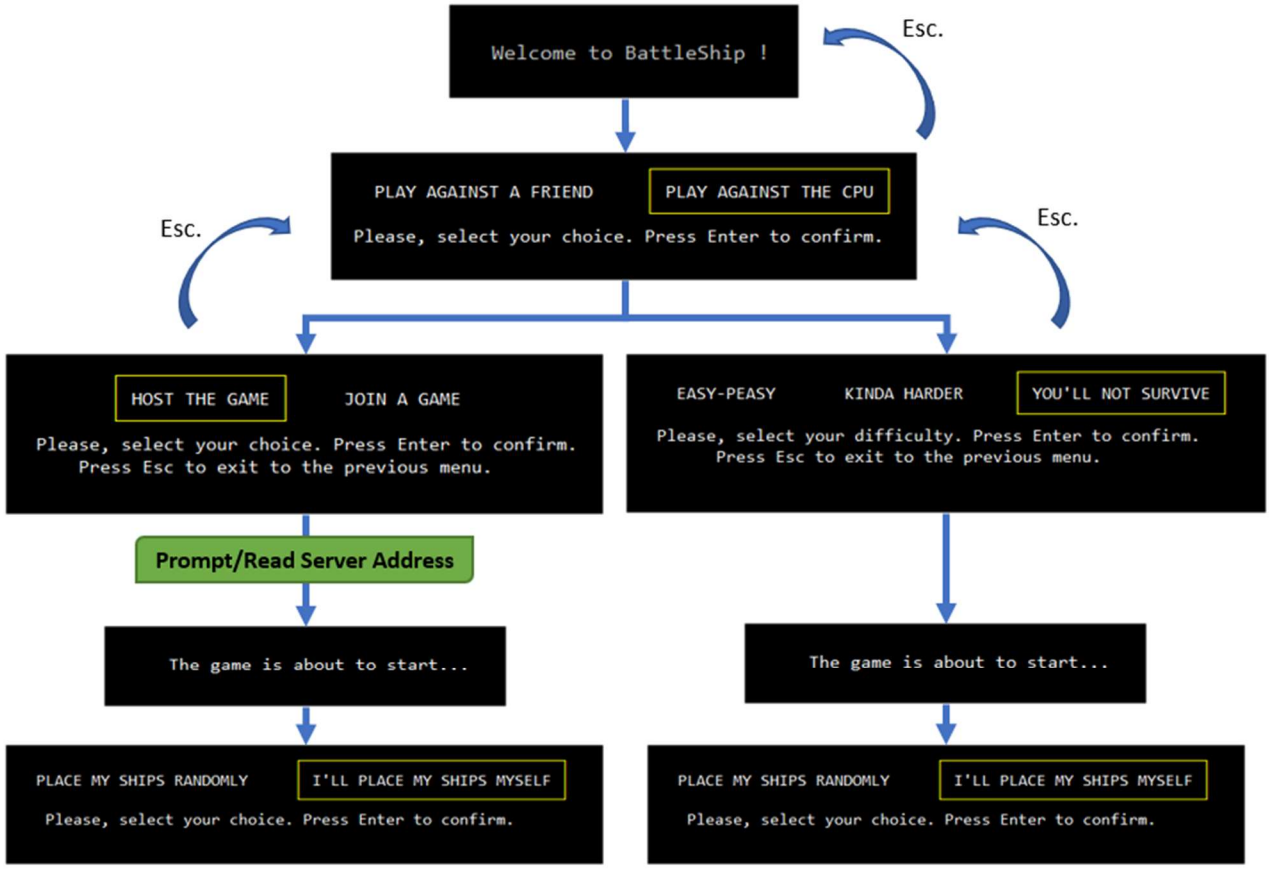
User Guide part. 1

When you start our Battleship game, a window appears to welcome you. Then you are confronted with a choice: to face the computer or to face another human player online.

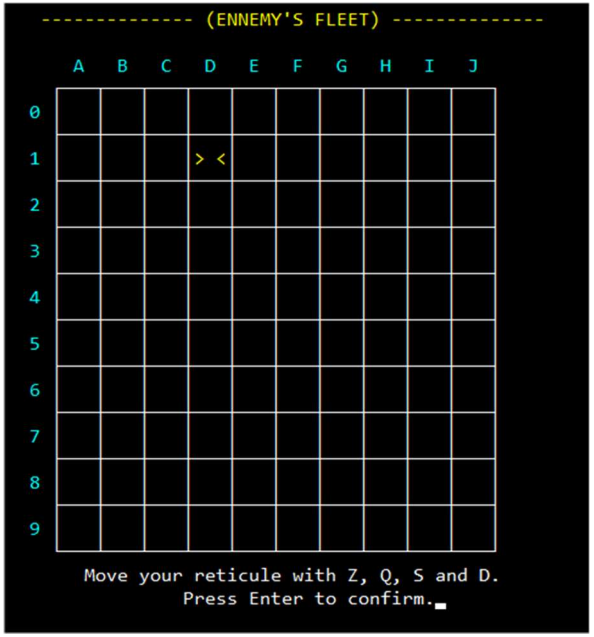
If you choose to play against a human player, you will have to make the decision whether to host the game, in which case the server will be created on your computer, or to join another player. In the first case, you will then see on the screen the address to which your friends should connect. In the second, a data field in which you will have to give the address of an existing server.

If you prefer to confront the artificial intelligence of the game, choose the option "Facing the computer". You will then have three possible difficulty levels, from Beginner to Expert. Choose one, and the game will start automatically as in two-player mode.

Once both players are ready (both humans or not), the game will start automatically. Once the game has started, it is no longer possible to go back with the 'Escape' button.

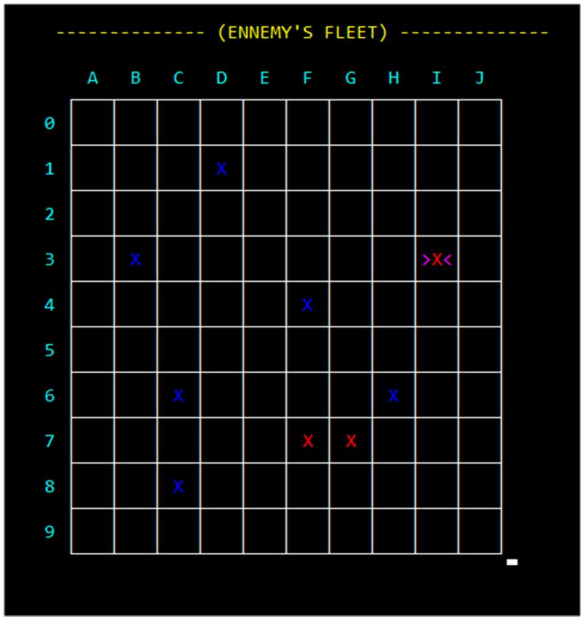


User Guide part. 2



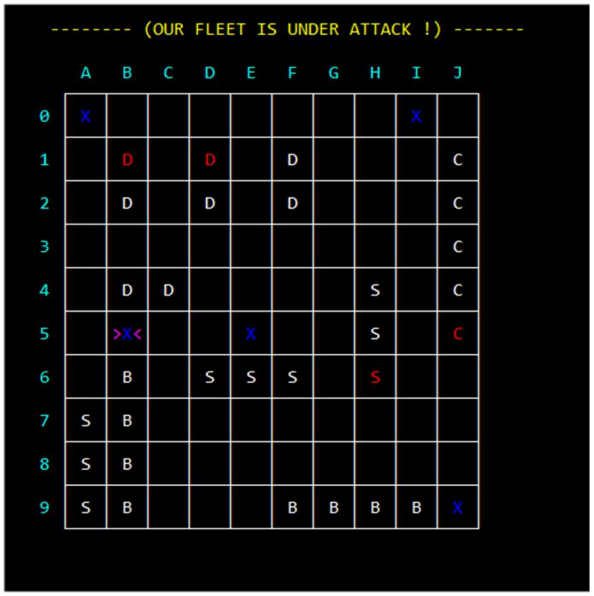
One of the pillars of BattleShip's game is to shoot at a grid representing a portion of the ocean. On this grid are hidden boats (invisible to the shooter), which are revealed when they are shot.

In our version of the game, the player who wants to shoot (see screen opposite), must move with the directional keys a crosshairs cell by cell to his target. By pressing Enter, he validates his choice and sends the shooting information to the server so that it can be transmitted to the target player.



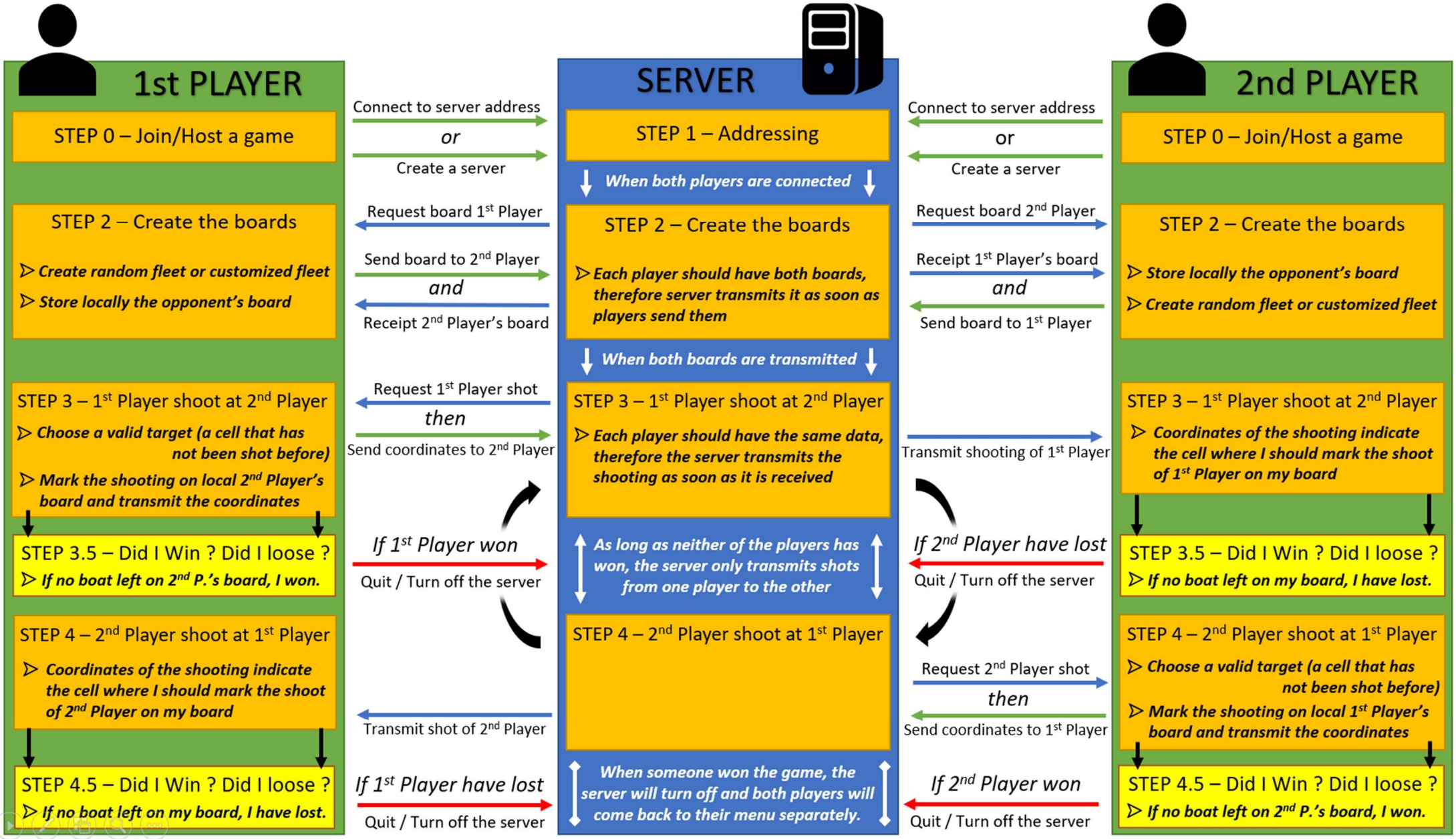
At the beginning of the game, the grid is completely empty except for the cursor that indicates the target cell. However, as the game progresses, the shooting screen will fill up to indicate the shots already made (cf. the grid to the right).

If the cell on which the player shot did not contain any boats, then it will be marked with a blue 'X'. If the cell contained a ship, then the 'X' will be red, regardless of the type of ship. Some similar games choose to display whether the affected boat is a submarine, aircraft carrier, etc. We believe that this distorts the principle of BattleShip and that this practice should be abolished.



When a player shoots you, in our game, you can see his shot live.
Thus, each player has, in addition to his shooting grid, a grid on which he can see his fleet and the shots fired by his opponent. On this grid, the boats are represented by a capital white letter according to their type (C for Cruiser, D for Destroyer, etc.) If the opposing player has shot into the water, the location will be marked with a blue cross. If he has fired on one of your ships, the letter in the affected cell will be marked in red. If at some point, you no longer have any white letters in your grid, you have lost the game.

Communication between players and server



Glossary of functions

```
short printBoard(char** matrice, short nb_colonnes, short nb_lignes, short votreboard);
short printBoard_af_tir(char** matrice, short nb_colonnes, short nb_lignes, coordonnees_tir dernier_tir, short votretir);
```

printBoard() and printBoard_af_tir() are both display functions. This means that they have a very important purpose of translating a grid of char into something visually attractive. The majority of our processes used the grids of char, but we did not want our players to manipulate something as ugly as the first board (see below). We wanted to add colours, and to be sure that our game is intuitive and well designed. Therefore, we created functions to transform our grids when we must display them. These functions used the extended ASCII table to print the grids.

0	0	0	X	C	0
X	0	B	0	C	0
0	0	B	0	c	S
B	0	0	0	c	s
B	0	X	0	C	s
0	0	0	0	X	X

REAL CONTENT OF THE GRID

Generally used for calculus or processing. Letters are for ship, X for opponent shootings when it arrived in an empty cell (marked as 0, it represents water)

using printBoard()

			X	C	
X		B		C	
		B		C	S
B				C	S
B		X		C	S
				X	X

WHAT FIRST PLAYER CAN SEE

Not often used in our code. This function is the first we create, and we derived a few tools from it.

0	0	0	X	C	0
X	0	B	0	C	0
0	0	B	0	c	S
B	0	0	0	c	s
B	0	X	0	C	s
0	0	0	0	X	X

REAL CONTENT OF THE GRID

Capital letters are used to indicate that the cell has not been shot yet. For example, a player can fire at a capital letter but not at a small one, because if it is small, it means that he already hit that cell once.

using printBoard_af_tir()

			X		
X					
				X	
				X	X
		>X<			X
				X	X

WHAT SECOND PLAYER CAN SEE

Called after each shoot that either player make. Its parameter "votretir" can modify the result to print the ships or to hide them. The brackets are given by dernier_tir, which indicates which cell has been shot at last.

The example just below is taken from the code of the function "printBoard_af_tir()". It shows how the contents of each cell are displayed. A loop runs through the original table and reads each square. A switch then allows you to decide what to display instead of the original cell. Depending on the case, you can choose to change the color of the letter, or to display it in capital. If the player is not supposed to know the exact content of the cell (because he has shot at it for example), then an X will be displayed instead. Each cell in the table is treated in the same way. Then you have to manage the ASCII characters and line display, but with time and work, you get a much more satisfying result than a traditional grid. The color() function is likely to change in the version of the final code, which will be ported to the Linux environment. However, the functioning of these remains unchanged.

```
case 'C' :
    if(votretir) printf(" ");
    else { color(15, 0); printf("C"); color(15, 0); }
    break;
case 'c' :
    color(12, 0);
    if(!votretir){ printf("C"); }
    else printf("X");
    color(15, 0);
    break;
```

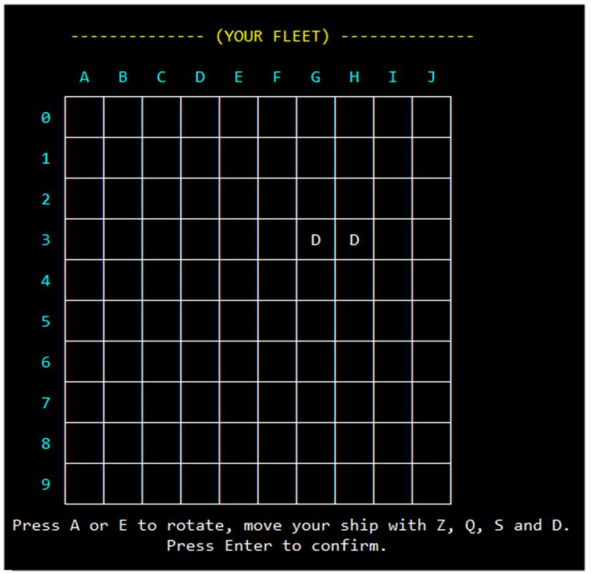
The functions creerMatrice() and libererMatrice() are very useful. They allow you to create dynamically allocated grid and to free it after you used it. By using a void type, we could easily cast the grid we create to be made of char or to be made of short values, since void, char and short types need all 1 byte of memory, and since we will never have more than 128 columns nor lines.

```
void** creerMatrice(short nb_colonnes, short nb_lignes);
void libererMatrice(short** matrice, short nb_lignes);
```

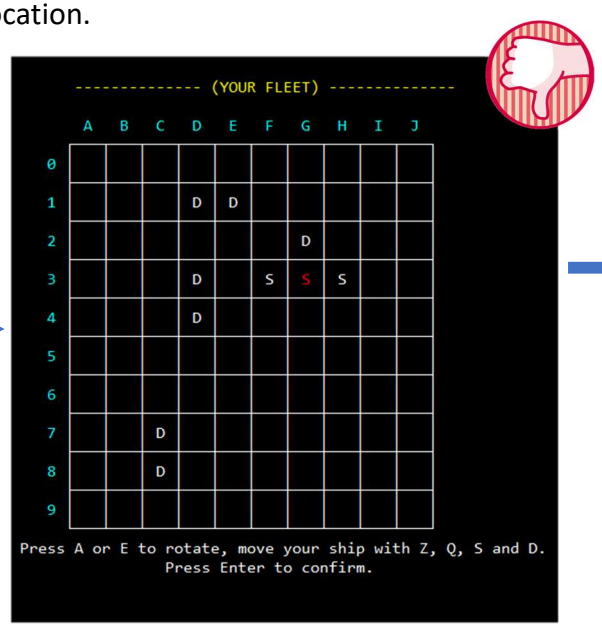
The programming language C is one of the only languages we know of that allows us to manage the allocation of variable memory in the program to such an extent. This asset is clearly essential for anyone who wants to claim to be a computer scientist. Using an int for a variable that will never exceed 128 is unworthy and unprofessional. We decided to code well, to code beautiful, to code clean. It is irresponsible to use variables unsuitable in C language, when it is one of its main characteristics to allow total control over memory and variable types.

```
char** placeShip(char** matrice, short nb_colonnes, short nb_lignes, short size_ship);
char** place_randomly(char** matrice, short nb_colonnes, short nb_lignes, short nb_ships);
```

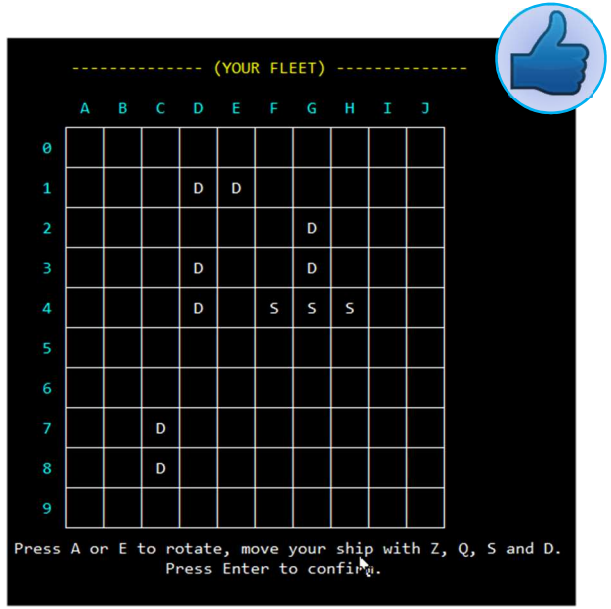
The functions presented above are very important. The first, placeShip(), allows to launch a protocol that asks the user to manually place a boat of size "size_ship" on the matrix passed as argument. In other words, if this function is called with matrix_P1 and size_ship = 2 as arguments, then Player 1 can move a two-cell boat on the grid until he wants to validate its location.



Player is free to place ships wherever he wants…



Except if it overwrites a cell that he filled in before.



There ! He must choose another place to validate.

Once all warships are placed, the game can begin. However, it is also possible to ask the computer to place the entire fleet for you! In this case, instead of calling the placeShip() function, we will use the place_randomly() function instead. This function automatically places the boats on the grid under the same conditions as the players. Positions and orientation of the ship are determined randomly.

```
coordonnees_tir Aim_randomly(char** matrice, short nb_colonnes, short nb_lignes, short difficulte);
```

Aim_randomly() also take a random column and a random line and return them as coordinates. This function is uniquely called to determine where the computer is going to shoot.

As you may have seen in the menu, this game provides 3 different levels of difficulty concerning the CPU. In fact, our Aim_randomly() function is a bit more complicated than just a random one. According to the difficulty you choose, you have one chance that the CPU fire at one of your boats instead of completely randomized. In fact, if you choose the max difficulty, for example, one time out of two, the computer will fire at one of the cells which contain one of your ship. The other time, it will just fire randomly, no matter if the cell is empty or not. Of course, neither a player nor a CPU can aim at a cell he/it already fired at.

```
coordonnees_tir Aim(char** matrice, short nb_colonnes, short nb_lignes);
```

The Aim() function allows you to select a cell that you did not fire at before. You can move your reticule with Z, Q, S and D keys, and validate your target using Enter. You will see immediately the result of your action by the call of printBoard_af_tir() function.

It can be important to notice that every player has the full grid of the other in local memory. This way, the player does not have to ask the server to ask the other player what is there in the cell he just shoot at. The system already knows it and it just has to transfer the line and the column of the shooting to the other player.

It is also important to highlight that we did everything by ourselves. This board, any boards in fact, the menu, everything is made out from scratch by our efforts, and not picked up from the internet or something. So that may not be perfect, but at least that is ours.