[update 18th Dec 2020 – 10:20am]

P-Project
Battleship

Battleship is a board game that mixes guessing and strategy. You first guess the position of the ships of your opponent. Once you hit a ship, you may follow specific strategies to cause the most damage to your opponents' fleet. It's not clear whether a strategy can take place before the 1st hit and it can be part of the fun to figure this out.

***Figure 1**. Battleship board game designed by Hasbro Gaming*

In this game, each player has a fleet composed of 28 warships. The ships are placed on a board following coordinates of columns and lines. **The goal is to hit your opponent's ships until no ship is left. The winner is the last one standing alive or the one with more points after the playing time is elapsed.**

Fleet's composition

The fleets composition of each player is the following:

Class of ship	Size	#Qtty
Carrier	5	2
Battleship	4	3
Destroyer	3	5
Super Patrol	2	8
Patrol Boat	1	10

The interface we presented in the previous section was built taking into consideration this composition for the fleet. Below you can see the same interface with the number of ships of each class (the Super Patrol were tagged as "SP #" and the Patrol Boat received only a number).

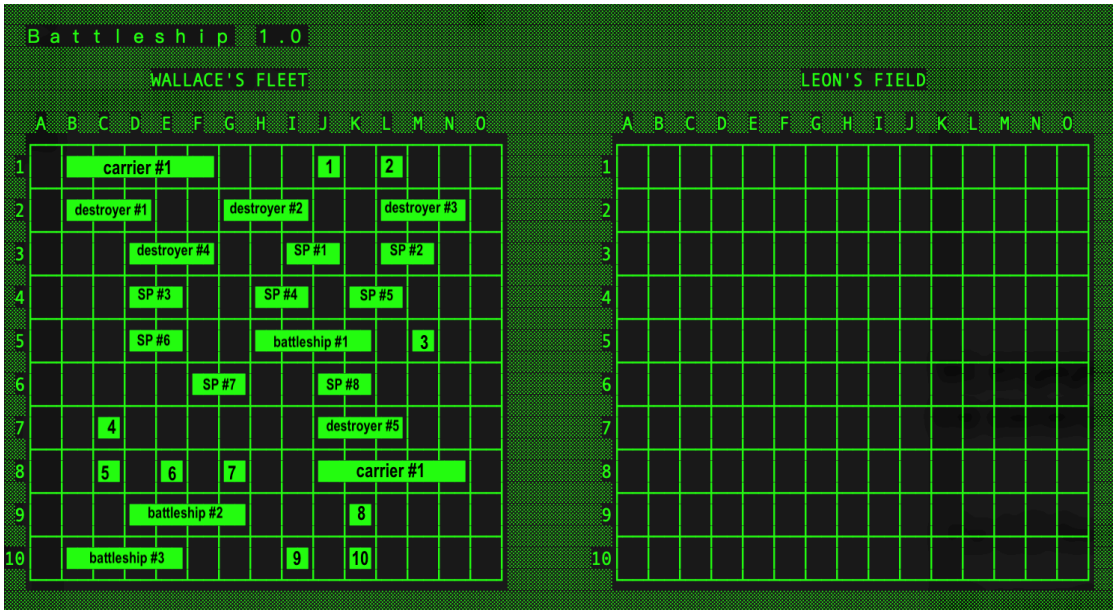

***Figure 3**. Fleet disposed in the player field. Opponent's field remains hidden.*

We hope that with such a large number of warships, the game will be engaging and challenging. While some big ships, like the carrier, present an opportunity to apply strategies after the 1st hit, small boats like the Patrol Boat may be tough to be hit by the missiles.

SHIP Placement: the basic version of the game has the ships placed randomly and automatically on the board. It's up to you to decide whether to place them only horizontally or include vertical positioning. No extra points for vertical positioning. **A feature for ship placement by the user is eligible to 0.5 extra points**.

Game Rules & Features

Modes Single and Multiplayer

The game must allow the players to choose against a virtual player (the so-called “computer”) or another human player with a client connected to the same server. Alternatively, the game may be developed for more than two players, as discussed in the Bonus Points session.

Game Start and Duration and Winning Rule

The players use their clients to connect to a chosen server that shares the same protocol. In its basic mode, once a client connects, it waits for the next client to connect and the pairs are matched immediately. The server must remain capable of receiving new connections and match new pairs simultaneously. Optionally, the game may support a match among more than two players, as described in the bonus points session. A game ends when the first of the two following conditions is true: (1) all the ships of one player were destroyed (a ship is destroyed when it was hit in all the boxes it occupies), or (2) the match reaches **5 minutes of duration**. The winner in condition 1 is the player with at least one remaining ship (the other with no ships left). When the game is finished due to condition 2, then the winner is the one that has more points. If a match ends by condition 2 and both players have the same number of points, then the winner is the one that destroyed more ships of the inferior classes (the hardest ones). If both players have the same points and destroyed the same number of ships of each class, then the match is finished with a **draw**. **UPDATE**: The game must have a panel showing the score of each one of the players (for example, like in Figure 4).

Turn-Taking

The game is of the “turn-taking” type, which means that players don’t fire their missiles at the same time, but in turns. To fire a missile, a player must guess a coordinate on the opponent’s field that [the player believes] has a ship to be hit. If a ship is hit (**hit event!**), then the player gets 1 point and the opportunity to attack again. The player remains firing missiles until an event “miss” happens (**miss event**). It’s considered a “miss” if the coordinate guessed by the player has no ship on the opponent’s field. **Each player has up to 30 seconds to guess a coordinate and fire a missile**. After this time, the opponent’s turn is started.

Scoring Rules

Players accumulate 1 point every time they hit one of the opponent's ships. When an opponent's ship is destroyed, the player receives an extra point. This way, for instance, destroying a carrier will give you 6 points (1 point per hit + 1 point for destroying the ship). The Patrol Boat is destroyed with only one hit and gives automatically 2 points (1 for the hit and 1 for destroying the boat).

Optional Rule: The Radar

ATTENTION: successful implementation of this feature gives 1 (one!) extra point to the grade of the P-Project for the pair.

You may, optionally, implement a radar. The radar consists of revealing to a player a portion of his/her opponent's field. The player (human or computer) gives a central coordinate and the game reveals the contents of the cells around it. This is illustrated in Figure 4.

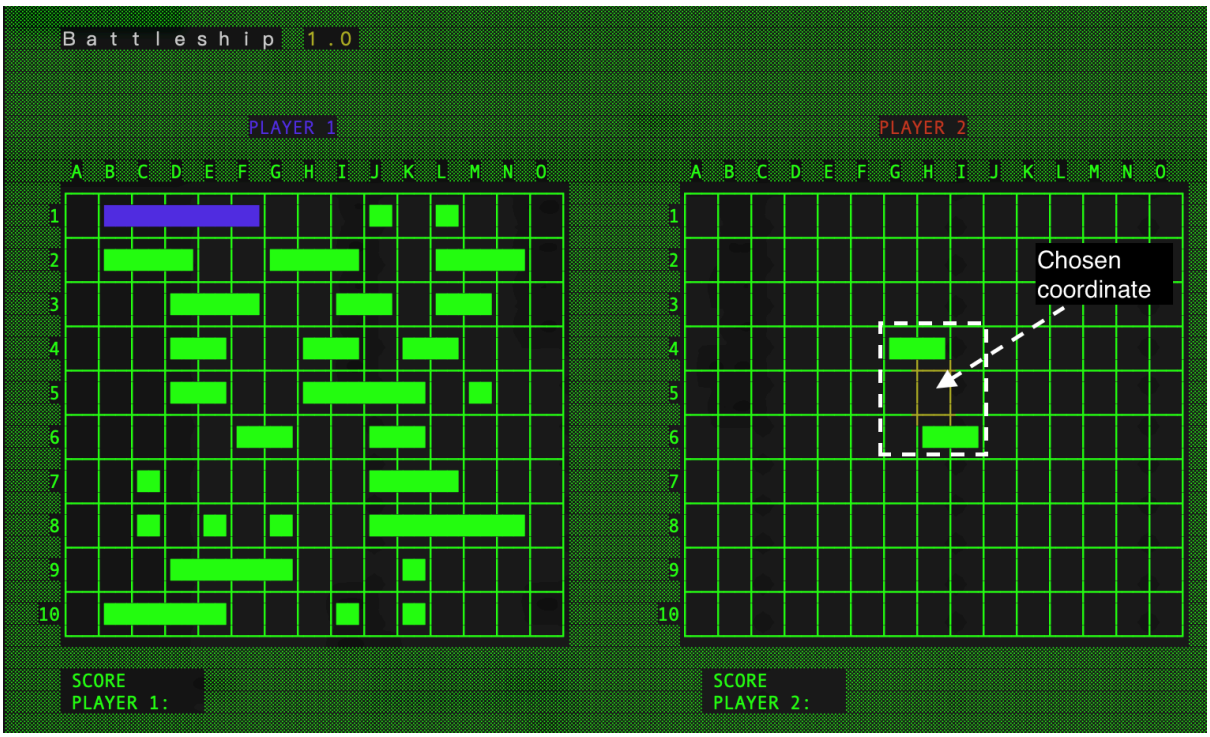

***Figure 4**. Illustrative Example of the RADAR feature*

If you decide to implement the radar feature, you must implement it both on the client and on the server-side. The radar feature, its presence or absence, must be signalized in the protocol you will define for the communication between the clients and the servers. This is part of the "hand-shaking" that happens at the beginning of each connection between client and server. The feature is automatically enabled when the clients of both players implement the radar. The radar round is triggered after every 4 rounds (1 round means each player has fired a missile or timed out) and becomes available for both players.

Extra Feature: The Lobby

ATTENTION: successful implementation of this feature gives 0.5 extra points to the grade of the P-Project for the pair.

You may, optionally, implement an extra feature for the game: the lobby. The lobby consists in showing a list of connected clients to the same server and allow clients to invite each other for a competition.

The implementation of the lobby implies defining special rules for the protocol. The clients that implement the lobby can receive invitation for competition and are shown on screen with some distinctive visual. The clients that do not implement the lobby will wait for the 1st pair to be formed to start the battle.

Other Opportunity for Extra Points

Tournament (0.5 - 1 points): a competition among the computer players developed by each pair. The winner is awarded 1 extra point and the 2nd place is awarded 0.5 points.

Best Protocol (0.5 points): each mentoring group will develop its own protocol. The protocols will be judged by a team of mentors, TAs and teachers. The mentoring group that develops the best protocol will be awarded 0.5 extra points on the p-grade (all members).

Chatbox (0.5 points): The successful implementation of the Chatbox entitles the pair to 0.5 extra points.

GUI (0.5 points): The pair who develops this game using a GUI will be awarded 0.5 extra points.

Game Mode for more than 2 playes (0.5 points): The pair who develops a multiplayer (more than two) game mode will be awarded 0.5 extra points. **UPDATE**: The game with more than 2 players is like the game with 2 players, but each player fires two missiles (one to each opponents). When Player A shoots on Player B, she/he sees the field of Player B, when shooting the C, sees the field of C. **All players must know the score of all other players**. A ship of Player B damaged by Player A does not heal for the round with Player C. Also, there's no rule to avoid players B and C of making a "non-aggression pact" and shoot only against Player A (for instance, they let 30s pass and time out their shooting opportunity against each other, or 'shoot on water').

Ship Placement (0.5 points): The basic version of the game has the ships placed automatically in a random position. The pair who develops a feature for ship placement by the user will be awarded 0.5 extra points.

Appendix: Implementation TIPs

Internals Matter

The Internals of the software, including the modularization, the correct use of the architectural pattern MVC, the documentation, the definition of the "contract" for each method and the correct implementation of each are the elements that make the most difference on your grade. A simple interface that reprints at each round will get the job done and make no harm to your grade.

TUI: Textual User Interface

The most simple interface that we can implement for this game is text-based. In its most simple mode, the game reprints the screen at each turn. Although simple, it is possible to create quite exciting interfaces using ASCII-Art. Some websites, like https://fsymbols.com/draw, allow you to draw your interface and then copy the text to the clipboard. The screen below was created with the support of the fsymbols.com website and adapted to a Java Class:

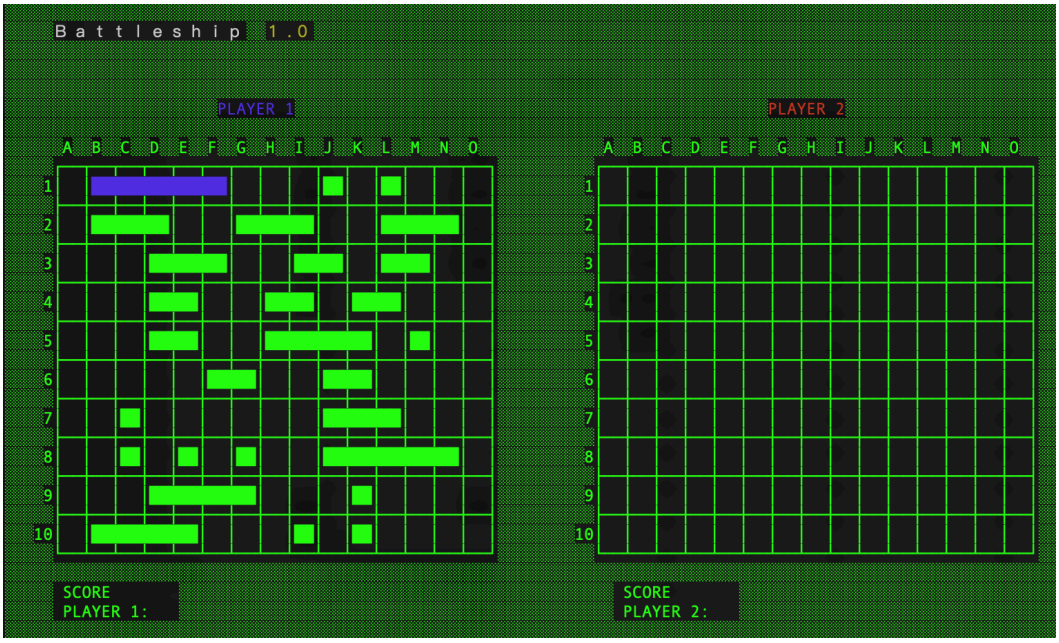

***Figure 2**: Example of Interface drew with ASCII Art*

Optionally, you can make it even more enjoyable by simulating an update on the screen. Usually, when we print new content to the standard output (like when we use `System.out.`[`println` or `print`] commands), the cursor is moved to the end of the text that was just printed. When we use "`println()`", it moves the cursor to the beginning of the next line. The trick here is to send special characters to the standard output that make the cursor "walk back". These characters are usually added to the end of the String. There are a few combinations of special characters you can use:

the carriage return ("`\r`"): moves to the beginning of the current line

the special combination ("`\033[F`"): moves to the previous line

the combination of both ("`\033[F\r`"): moves to the beginning of the previous line

Once the cursor is positioned where you want, you can use "`System.out.print(…)`". The content you print will replace the content currently presented on screen. In this example, I added 28 times the third combination to the String that prints the board. This way, after printing the board, the cursor is back to line 1.

TUI: Playing with colors

Additionally, you can also use colors to improve the *look'n feel* of your game interface. The following Java code has special combinations of characters you can use for playing with colors on the termina (next page).

To print characters in a specific color, you have to first print the special combination of the color you want. Then you print the content you want in this color. Finally, you print the "RESET" combination. See an example below:

```
System.out.println(TerminalColors.RED + "This text is RED!" + TerminalColors.RESET);
```

TUI: Enjoy It

If you are engaged and motivated to work on a fancy TUI (or even a GUI) and you feel like you and your partner can make it without harming the quality of the other parts of your software, then do it. Have fun with the project. Feel proud of it. We give these tips to the students who want to make something more and are proud of having their first software developed. However, an interface that reprints and scrolls at each interaction (like the Hotel Application) is simple enough to be made and will get the job done without harming your p-project grade. Think about it and make a wise decision. Start by prioritizing your tasks and project decisions.

```java
public class TerminalColors {
    // Reset
    public static final String RESET = "\033[0m";  // Text Reset

    // Regular Colors
    public static final String BLACK = "\033[0;30m";   // BLACK
    public static final String RED = "\033[0;31m";     // RED
    public static final String GREEN = "\033[0;32m";   // GREEN
    public static final String YELLOW = "\033[0;33m";  // YELLOW
    public static final String BLUE = "\033[0;34m";    // BLUE
    public static final String PURPLE = "\033[0;35m";  // PURPLE
    public static final String CYAN = "\033[0;36m";    // CYAN
    public static final String WHITE = "\033[0;37m";   // WHITE

    // Bold
    public static final String BLACK_BOLD = "\033[1;30m";  // BLACK
    public static final String RED_BOLD = "\033[1;31m";    // RED
    public static final String GREEN_BOLD = "\033[1;32m";  // GREEN
    public static final String YELLOW_BOLD = "\033[1;33m"; // YELLOW
    public static final String BLUE_BOLD = "\033[1;34m";   // BLUE
    public static final String PURPLE_BOLD = "\033[1;35m"; // PURPLE
    public static final String CYAN_BOLD = "\033[1;36m";   // CYAN
    public static final String WHITE_BOLD = "\033[1;37m";  // WHITE

    // Underline
    public static final String BLACK_UNDERLINED = "\033[4;30m";  // BLACK
    public static final String RED_UNDERLINED = "\033[4;31m";    // RED
    public static final String GREEN_UNDERLINED = "\033[4;32m";  // GREEN
    public static final String YELLOW_UNDERLINED = "\033[4;33m"; // YELLOW
    public static final String BLUE_UNDERLINED = "\033[4;34m";   // BLUE
    public static final String PURPLE_UNDERLINED = "\033[4;35m"; // PURPLE
    public static final String CYAN_UNDERLINED = "\033[4;36m";   // CYAN
    public static final String WHITE_UNDERLINED = "\033[4;37m";  // WHITE

    // Background
    public static final String BLACK_BACKGROUND = "\033[40m";  // BLACK
    public static final String RED_BACKGROUND = "\033[41m";    // RED
    public static final String GREEN_BACKGROUND = "\033[42m";  // GREEN
    public static final String YELLOW_BACKGROUND = "\033[43m"; // YELLOW
    public static final String BLUE_BACKGROUND = "\033[44m";   // BLUE
    public static final String PURPLE_BACKGROUND = "\033[45m"; // PURPLE
    public static final String CYAN_BACKGROUND = "\033[46m";   // CYAN
    public static final String WHITE_BACKGROUND = "\033[47m";  // WHITE

    // High Intensity
    public static final String BLACK_BRIGHT = "\033[0;90m";  // BLACK
    public static final String RED_BRIGHT = "\033[0;91m";    // RED
    public static final String GREEN_BRIGHT = "\033[0;92m";  // GREEN
    public static final String YELLOW_BRIGHT = "\033[0;93m"; // YELLOW
    public static final String BLUE_BRIGHT = "\033[0;94m";   // BLUE
    public static final String PURPLE_BRIGHT = "\033[0;95m"; // PURPLE
    public static final String CYAN_BRIGHT = "\033[0;96m";   // CYAN
    public static final String WHITE_BRIGHT = "\033[0;97m";  // WHITE

    // Bold High Intensity
    public static final String BLACK_BOLD_BRIGHT = "\033[1;90m"; // BLACK
    public static final String RED_BOLD_BRIGHT = "\033[1;91m";   // RED
    public static final String GREEN_BOLD_BRIGHT = "\033[1;92m"; // GREEN
    public static final String YELLOW_BOLD_BRIGHT = "\033[1;93m";// YELLOW
    public static final String BLUE_BOLD_BRIGHT = "\033[1;94m";  // BLUE
    public static final String PURPLE_BOLD_BRIGHT = "\033[1;95m";// PURPLE
    public static final String CYAN_BOLD_BRIGHT = "\033[1;96m";  // CYAN
    public static final String WHITE_BOLD_BRIGHT = "\033[1;97m"; // WHITE

    // High Intensity backgrounds
    public static final String BLACK_BACKGROUND_BRIGHT = "\033[0;100m";// BLACK
    public static final String RED_BACKGROUND_BRIGHT = "\033[0;101m";// RED
    public static final String GREEN_BACKGROUND_BRIGHT = "\033[0;102m";// GREEN
    public static final String YELLOW_BACKGROUND_BRIGHT = "\033[0;103m";// YELLOW
    public static final String BLUE_BACKGROUND_BRIGHT = "\033[0;104m";// BLUE
    public static final String PURPLE_BACKGROUND_BRIGHT = "\033[0;105m"; // PURPLE
    public static final String CYAN_BACKGROUND_BRIGHT = "\033[0;106m";  // CYAN
    public static final String WHITE_BACKGROUND_BRIGHT = "\033[0;107m";   // WHITE

}
```