[updated 13th Jan 2021 – 00:40am]

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).

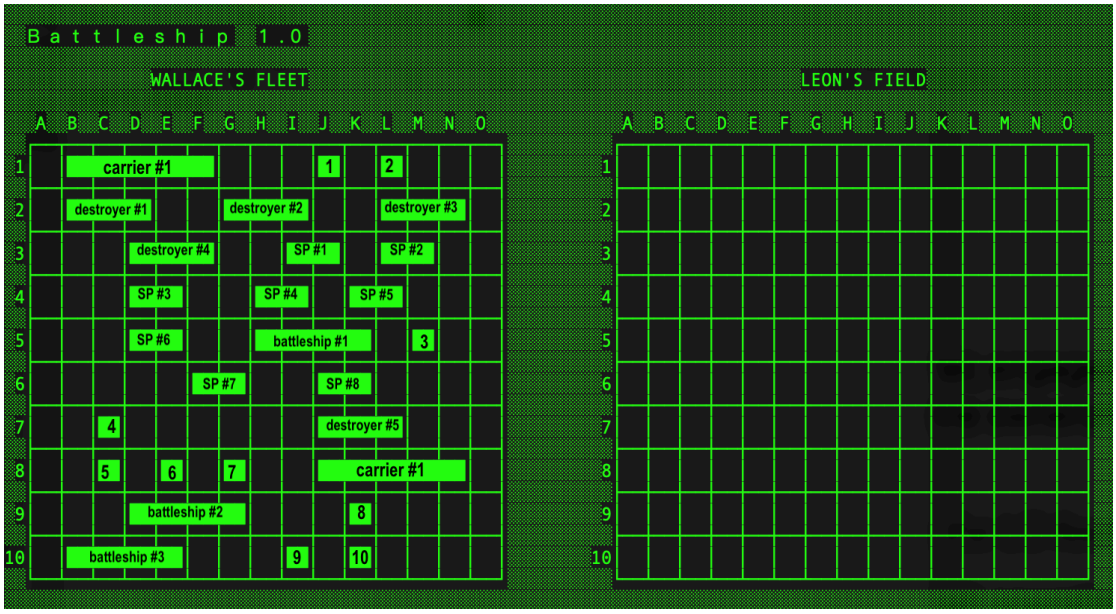

***Figure 3**. Fleet disposed in the player field. Opponent's field remains hidden.*

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

Server Requirements

Requirement S01: the server must have a default port for listening to connections. However, it must be possible to inform a different port.

Requirement S02: the server must enforce unique names among the players. When a client connects to the server, it must inform the player name. If the name is existing, the server must inform the client that the name is already in use, forbid the connection. **TIP**: this can be part of a connection procedure, like a “handshake.” (not represented in the process below, in BPMN).

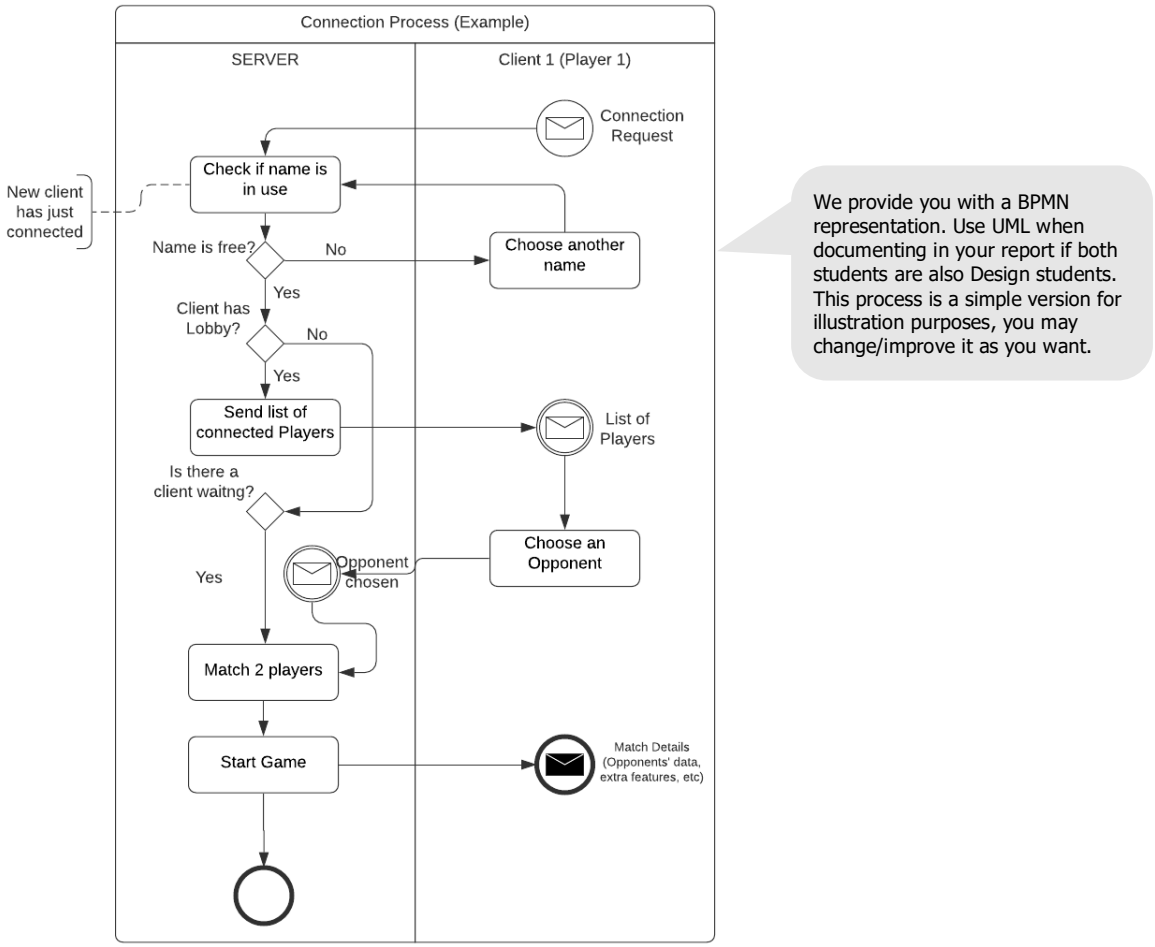

***Figure 4**. Example flow (in BPMN) of the Client/Server Connection Procedure*

Requirement S03: the server must allow multiple pairs of clients to connect and play. In its simplest version (without the Lobby), a client is matched with the next client that connects to the same server.
If the Lobby is implemented, for both the client and the server, then the server must send a list of connected players to the client. The client chooses an opponent, the opponent is notified and may accept or not the challenge. Once the challenge is accepted, the two players (two clients) start a match.

TIP: Upon a new client connection, the server checks if the client has the Lobby implemented. It can be asked to the client or, if you prefer, the protocol itself may inform this by the version number, for instance.

- *If the Lobby is implemented in the client, you can send a list of players so that the client can 'challenge' them.*
- *If the Lobby is not implemented in the client, then you have to think of a strategy. For instance, you can put the clients with no lobby in a list to be challenged by someone with Lobby and/or match the 'non-lobby' client with the next 'non-lobby' client that connects.*

Requirement S04: when a player makes a move (fires a missile), the server must allow it even if the client already fired a missile in that coordinate. If a ship was hit, it won't be hit again. The server must inform the client whether a **hit event** or a **miss event** occurred. In case of a hit, Server must update the score of the player in its own state and update all players of the match of the new state. The points awarded must follow the game rules (1 point per hit + 1 extra point when a ship is entirely destroyed).

NOTE: to improve the quality of your client for the players, the client **must** forbid the players to fire missiles to coordinates that were already tried. Optionally, it may present the opponents field in its UI, illustrating the events (hit or miss) to the player.

Requirement S05: if a client disconnects during a match for any reason (a bug, for instance), the server must continue to run 'smoothly' and inform the other player that the match was won by Walkover (WO).

Requirement S06: if a client takes more than 30s to fire a missile, the server must notify the client and give the turn to the other player.

Client Requirements

Requirement C01: the client must be implemented following the MVC Pattern.

Requirement C02: at the initialization, the client must ask to the player: the IP and port of the server and the name of the player.

Requirement C03: the client must be able to receive a message from the server indicating whether the chosen name is available or not. If not available, it must prompt the player and ask for a new name.

Requirement C04: the client must be able to randomly place the ships of its player and also show their state. The state must be updated on screen after every state change in the server. Horizontal ship placement is mandatory. Vertical is not (no extra points).

Requirement C05 (optional, extra feature): the client may allow the player to place their ships. This requirement is optional and is considered an extra feature that awards extra points.

Requirement C06: the client must control the time for the player to fire a missile. The player must fire a missile within 30s. In the case the server notifies the client of the timeout, then the client must cancel the user input request and wait for the move of the other player.

Requirement C07: the client must show the scores of all the players in the match.

Requirement C08: the client must notify the player of a miss event or a hit event related to the player action or related to the opponent's action. In case the client receives notification of a hit event on one of its ships, it must update the state on its user interface. **Optionally**, the client may keep and update a visual representation of the hits and the misses on the **opponent's field**, this way the player has better information to fire the next missile.

Requirement C09: the client must notify the player of a win or a loss and the type of the victory (WO or Scores).

Requirement C09: the client must forbid the player of making illegal moves. Illegal moves are:

- Firing missiles outside the coordinates of the field (15 columns x 10 rows).
- Firing missiles on coordinates that were already tried.

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, both the server and the client wait for the next client to connect. The server matches pairs immediately. The server must remain capable of receiving new connections and match new pairs simultaneously, so that multiple games can be played at the same time.
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**. **IMPORTANT**: The game must have a panel showing the score of each one of the players (according the requirement C07. There’s an example illustrated in Figure 5).

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** (Requirements S06 and C06). 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).

Extra Feature: 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.

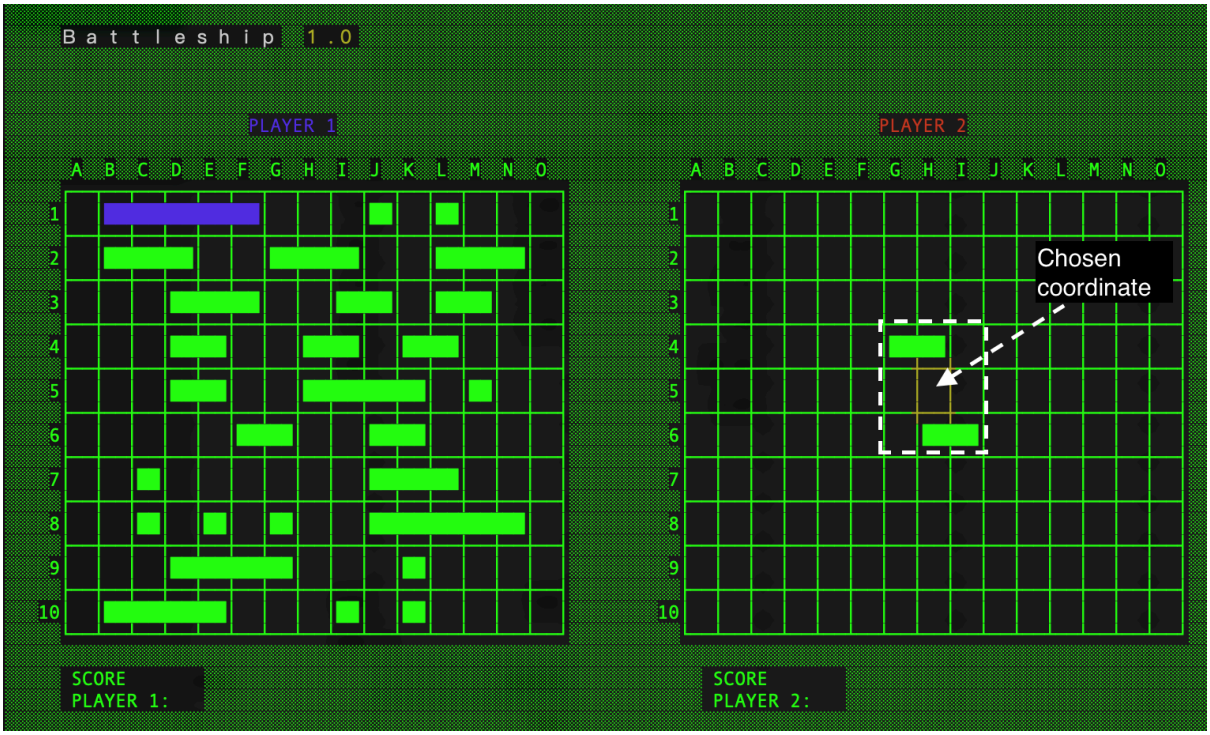

***Figure 5**. Illustrative Example of the RADAR feature*

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 (See example in Figure 4). 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 Opportunities 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. (Note: all clients and server must be in the same network or you have to manage the Firewall and port-forwarding rules to make it work).

Radar (0.5 points): The successful implementation of the Radar entitles the pair to 0.5 extra 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 players (0.5 points): The pair who develops a multiplayer (more than two) game mode will be awarded 0.5 extra points.
NOTE: 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.

Report Requirements

The report must be written in readable English. The *target audience* of this report is a *software maintainer*, i.e., somebody who did not necessarily write the code himself, but at a later occasion will have to extend or improve it. Therefore, this person should be able to understand the overall design of your application, understand the purpose and functionality of each class, be able to check whether the application passes at least the existing tests, etc.

Discussion of the Overall Design Your report should discuss the overall design of the application (global structure of the system), in terms of the classes and their relationships. The overall design should consist of the following:

1. Class diagrams, with explanations of the global design and the roles and responsibilities of each package and class. Make sure your diagram layout reflects the structure of your application, e.g., by repositioning classes and removing unnecessary details. Feel free to add extra labels and notes. The class diagram should show all public methods and fields. Fields with a type that is contained in the class diagram should be represented as an association4.
2. A systematic overview of which functional requirements are implemented by which classes and methods. If you did not manage to implement all requirements, this should be mentioned here.
3. A description of how you implemented the *Model-View-Controller* pattern: indicate which classes and packages play the role of model, view and controller in your application design.

Testing Testing is an integral part of the development of an application. Therefore, in the report you are expected to discuss both unit tests and system tests. Also, if you discovered errors in your implementation during testing and had no time to fix these errors, you should mention them in this part of the report. Mention at least one discovered and fixed bug and include the test cases of this bug. **NOTE**: The Appendix A of the manual has an example of how to document your tests.

Unit Testing. All complex (non-trivial) methods should be tested with (at least) one of the following meth- ods, if possible:

- Using dedicated test classes, preferably using JUNIT, as you have learned in this module. This is the most thorough testing method for unit tests.
- Simulating a part of the system manually or programmatically via telnet, to test the communication over the network.
- Adding a main method that creates several instances of the class under test, and then invokes several methods on these objects.
- Visual inspection of the UI.

For each class, the report contains the following information concerning the unit tests:

1. Strategy used to test the class (in isolation, together with other classes, not at all);
2. Test method that has been applied (see above);
3. If applicable, test programs that have been developed;
4. Test results and expected results;
5. Test coverage percentage of each method to be determined using the Emma plugin. In case of low coverage, discuss reasons and consequences for this class.

The test report should provide sufficient information for the reader to repeat the tests. Test programs should be included in the delivered code.

System testing. System tests are used to assess the functionality of the application as a whole; i.e., a system test will typically consist of a complete run-through of a game. The functional requirements should serve as the basis for these tests, since the implementation should fulfil these requirements. Each requirement may be tested separately by creating one or several test executions with different usage scenarios (of both expected and incorrect usage). Your system tests should also be described in the report, indicating which aspects of the system have been tested and how. Appendix A.2 contains an example of how to document a system test.

Metrics report The report should contain a (brief) section that contains the values of the software metrics you have learned about during this module. Include an analysis of the consequences for the quality of the code5 .

Academic Skills Chapter In Week 7, you are asked to make a planning for the project, and to discuss this with a teaching assistant. During the last weeks of the module, you should keep track of how your planning corresponds with your actual progress, and adapt your planning if necessary. In your report you should reflect on this. In particular you should describe the following:

1. How was your planning influenced by your experiences with the planning and time writing during the Design project?
2. To what extent did your planning correspond to the actual progress during the project weeks? What made you deviate from your planning? For example:

    • Were there project tasks that took significantly more or less work than you had expected? If so, why did this happen?

    • Were there significant losses of time or momentum in your projects (one or more periods in which you just did not seem to make the progress you had intended)? Looking back, how do you explain this?

3. Which counter measures did you take to compensate for deviations from your original planning? What was the impact of this on the intended scope or quality of the project?

4. What did you learn from this experience for your next (project) planning? Take your answers to the other questions into account and ask yourself how you would want to prevent this or deal with this next time.

5. Suppose that next year you are a teaching assistant for this project. Give at least two do's and don'ts that you would tell your students to help them with their planning.

TIP: We produced a MS-Word Project Report Template (available on Canvas) that you can use to write your report.

Grading Criteria

The Grade of the Project is important part of your assessment in the Programming Study Unit. The project-grade is an average of the grade you receive for the report and the grade you receive for the implementation. Therefore, which makes the report very important to the composition of your project-grade.

To encourage timely submissions, groups that hand in the project before the early bird deadline (Wednesday 23:59 of Week~10) will get a bonus of 0.5 points on the total project grade. Pairs that miss the early bird deadline must hand in their submission no later than Friday (23:59) of Week~10. This is a hard, final deadline, i.e., missing it causes you to fail the module.

This document lists – in previous pages – several ways to earn bonus points, which are added to the average grade of the implementation. The bonus points are only added to the projects that obtained a pass grade (5.5) on the required features. The list of grading criteria comprises: Packaging, Global Design, Programming, Testing, Academic Skills Chapters, Report.

Packaging – Your submission must be handed in via Canvas according to the requirements defined in Packaging for Submissions (Canvas Page). Grading scale: fail/pass. *Pass required*.

The scale below will be used for all the criteria, except the ones only "fail/pass"

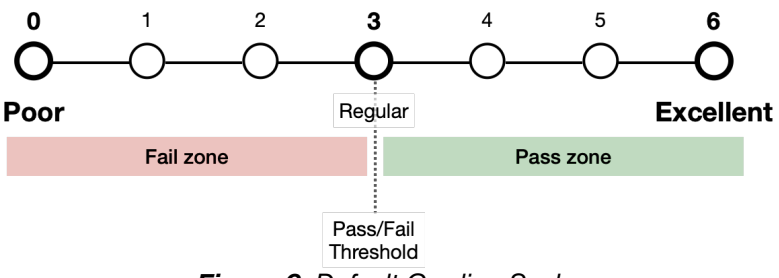

***Figure 6**: Default Grading Scale*

Global Design – The quality of the global design your project will be judged considering two criteria: (1) Overall design (Design and implementation) and (2) Class Specifications (programming by contract)

(1) Overall Design: consists of the assessment of the Design (class diagram, packaging of classes) and MVC implementation.
Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: (a) The implementation code is generally structured according to the *Model-View- Controller* pattern. In the report you elaborated on how you implemented the *Model-View- Controller* pattern. If certain functionality is not separated as it should, a description of the necessary changes is included in the report. (b) The program is divided into classes and pack- ages in a logical way. (c) A class diagram for all model classes is included and described in the report. **Pass required***.*

- *Excellent:* The implementation fully adheres to the *Model-View-Controller* pattern, and concrete classes are shielded by interfaces whenever applicable. The report elaborates on all specific design choices, and includes (not automatically generated) UML class diagrams for all classes. All functional requirements are defined and elaborated in the report.

(2) Class Specifications
Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: (a) The implementation code is generally structured according to the *Model-View- Controller* pattern. In the report you elaborated on how you implemented the *Model-View- Controller* pattern. If certain functionality is not separated as it should, a description of the necessary changes is included in the report. (b) The program is divided into classes and pack- ages in a logical way. (c) A class diagram for all model classes is included and described in the report. **Pass required***.*

- *Excellent:* The implementation fully adheres to the *Model-View-Controller* pattern, and concrete classes are shielded by interfaces whenever applicable. The report elaborates on all specific design choices, and includes (not automatically generated) UML class diagrams for all classes. All functional requirements are defined and elaborated in the report.

Programming – The quality of the code of your project will be judged considering four criteria: (1) Overall code quality, (2) Implementation of Functional Requirements (completeness), (3) Game Rules, and (4) Documentation

(1) **Code Quality**. Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: Names of classes, variables and methods follow the conventions prescribed in the module (checked with the Checkstyle plugin) and have intelligible names. Constants are used where applicable to make the software easier to maintain. The code is neat: not unnecessarily complicated, no excessive duplicate code is present and no code is unused. The coverage of the project is calculated and explained in the report. You will pass if most of these conditions are largely met. **Pass required**.
- *Excellent:* All of the above conditions are fully met. All software metrics you have learned about during this module are calculated, analysed and explained in the report.

(2) **Functional Requirements**. Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: A functioning multi-threaded application in which multiple games can be played simultaneously with your client/server implementation over the network. Your implementation adheres to the basic rules in the protocol. We specifically take the quality of the networking, multithreading and input validation into account. **Pass required***.*
- *Excellent:* Your implementation fully adheres to the set protocol, handles all networking exceptions well with custom exceptions and contains all configuration options mentioned in the functional requirements.

(3) **Game Rules**. Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: Your implementation partially complies with the game rules. **Pass required***.*
- *Excellent:*Your implementation fully complies with the game rules.

(4) **Documentation**. Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: Most classes, methods and tests are documented with minimal useful Javadoc tags (description + parameter definition). **Pass required***.*
- *Excellent:* All classes, methods and tests are documented with extensive and detailed Javadoc. Complex parts of code are documented with inline comments.

Testing

(1) **Unit Tests**. Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: The most complex methods of the three most complex classes are properly tested with JUnit. The most complex methods have 100% coverage. The chosen classes and test coverage are mentioned and justified in your report. **Pass required**.

- *Excellent:* All complex (non-trivial) methods that are tested with JUNIT tests, telnet, main methods or other unit testing methods, whenever possible. The tests and their coverage are discussed and justified in the report.

(2) **System Tests**. Grading scale: Poor-to-Excellent (Figure 6)

- *Pass*: The report describes how five functional requirements have been tested as discussed in Appendix A, or any similar reference. Examples of functional requirements are *"after a game is finished, the user can start a new game"* and *"a user should be able to input another port if a specified port is invalid"* from the *"All user input should be validated"* functional requirement. **Pass required**.

- *Excellent:* All implemented functional requirements are system tested and described in the report as mentioned above.

Academic Skills Chapter (Individual, 1 chapter per member)

(1) **Time Management Plan**. Grading scale: Pass-Fail

- *Pass*: A time management plan is presented and evaluated at the end of each week. Minimum of 3 weeks (Reference: you can make a shorter version of the Academic Skills Assignment regarding the weeks of the Project). **Pass required**.

(2) **Strengths and Weaknesses**. Grading scale: Pass-Fail

- *Pass*: The student individually lists her/his Strengths and Weaknesses considering the Topics of the Learning Journey (Programming Topics). **Pass required**.

(3) **Receiving/Giving§ Feedback**. Grading scale: Pass-Fail

- *Pass*: The student reflects on the peer feedback received and given. Necessary improvements are listed (if any). **Pass required**.

Report

(1) **Structure**. Grading scale: Pass-Fail

- *Pass*: The reader can quickly understand what the text is about, the text is logically structured and all necessary information can be easily found. **Pass required**.

(2) **Writing**. Grading scale: Pass-Fail

- *Pass*: The language in the report is understandable for the target group, spelling is correct and consistent and sentences are well-constructed. **Pass required**.

Extensions

(1) **Tournament**. 1 point (winner), 0.5 points (2nd place)
(2) **Best Protocol**: 0.5 points (for the whole mentoring group)
(3) **Chat box**: 0.5 points
(4) **Graphical User Interface (GUI)**: 0.5 points
(5) **Match among +2 players**: 0.5 points
(6) **Ship Placement**: 0.5 points
(7) **Lobby**: 0.5 points
(8) **Radar**: 0.5 points

Grade Composition and Rules for Passing

The Assessment of the Project is done with two types of scale: Fail/Pass and Poor-to-Excellent.

The assessment criteria graded with Fail/Pass represent constraints your project have to adhere in order to pass. These elements don't compose the numeric value of your final Project Grade. One special rule is that, for passing, you can have up to ONE (1) criterium graded as 'Fail'.

The assessment criteria graded with the 'Poor-to-Excellent' (this scale is illustrated in Figure 6, page 11) will compose your grade. Each element of the scale is translated into a number from 1 to 10 (this scale translation is illustrated in Figure 7, next page) and they receive different weights. After translating each grade from Poor-to-Excellent into numbers, these values are weighted according to the weight of each criterium. Overall Design and Game Rules have a weight of 1.5, Code Quality and Functional Requirements have a weight of 2.0. The other criteria have weight of 1.0. You can see a complete example in the Figure 7 (next page).

Project Grading Example

Student name - s-Number	Fail/pass Criteria					Graded Criteria										
			Academic Skills		Report	Global Design		Programming					Testing			
	Packaging	TMP*	Strengths & Weaknesses	Peer Feedback	Structure	Writing	Overall Design	Class Specs	Code Quality	Func. Req.	Game Rules	Docum entation	Unit Testing	System Testing		
Weigth			- - Non Applicable - -				1,5	1,0	2,0	2,0	1,5	1,0	1,0	1,0	total -->	11,0
xxxxx xxxxxxx - s999999	Pass	Pass	Pass	Fail**	Pass	Pass	Excellent (6)	5	Regular (3)	Excellent (6)	Excellent (6)	Regular (3)	4	4		
						Points from scale	10,0	8,5	5,5	10,0	10,0	5,5	7,0	7,0		
						Weighted Points	15,0	8,5	11,0	20,0	15,0	5,5	7,0	7,0		
												11	(weighted average)			
											Grade before bonus	89,0	÷	8,13		

* **TMP** stands for '**T**ime **M**anagement **P**lan'

** Fail: You're allowed to have the maximum of ONE (1) fail

Scale

Bonus Points	
Tournament	0
Best Protocol	0,5
Chat box	0
GUI	0,5
Match +2 players	0
Ship Placement	0
Lobby	0
Radar	0
Total Bonus Points	**1,0**

Final Grade **8,13 + 1,0 = 9,13**

***Figure 7**: Project Grading Example*

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:

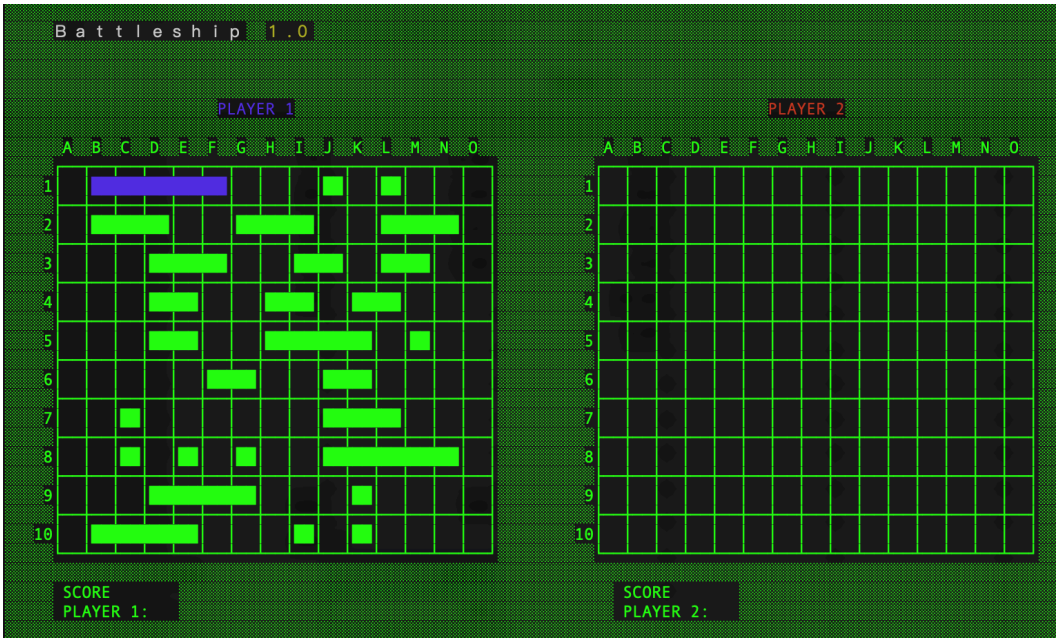

***Figure 8**: Example of Interface drew with ASCII Art*

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

}
```

Follow this link to download the Java File.