SPACE WARS

ALEJANDRO LÓPEZ
LAURA TORO
MONICA DOMINGUEZ

Institut Esteve Terradas
CFGS DAM
2014-25

Table of contents

Mónica Domínguez
Laura Toro
Alejandro López

1. Project Introduction

○ Brief Description of the Game

Space Wars is a video game where we control the army of a planet that is under continuous attack from enemy planets. By upgrading the planet's technology and purchasing new ships, we try to withstand the enemy waves that wish to conquer us.

The planet generates resources every minute, which are necessary to upgrade technology and buy ships. Additionally, we can earn these materials by winning battles.

○ Project Objectives

The main objective of this project is to incorporate and utilise the course's teachings into the real development of an application (in this case, a game). The more specific objectives are the following:

- Correct implementation of Object-Oriented Programming.
- Generation of a Graphical User Interface (GUI).
- Connection of the application to a remote database.
- Ability to extract data from the database and convert it into an HTML file, through an initial XML document that is transformed using XSLT.
- Creation of a website.
- Creation of a class diagram for our application.
- Proper use of Git for version control.

○ Used Tools (IDE, Programming Language, Libraries, Frameworks...)

A multitude of tools have been used to accomplish the fulfillment of the project. These tools are listed and classified below:

Application Programming

- Java → Programming language
- Eclipse → IDE (Integrated Development Environment)
- Java Swing → GUI library
- MySQL Connector (8.0.15) → Library for database connection

Database

- Amazon Web Services → Database hosting
- SQL → Language for creating and managing the database

Website

- HTML and CSS → Web page generation
- JavaScript → Enables interaction within web pages and is used to retrieve HTML report files and move them into a directory accessible by the website
- Visual Studio Code → IDE

Version Control

- Git → Version control tool (VCS)
- GitHub Desktop → Graphical tool for interacting with Git

Class Diagram

- Draw.io → Tool for creating diagrams
- Excalidraw → Tool for sketching diagrams

2. Analysis and Design

○ Functional and Non-Functional Specifications

A way to classify the specifications of our game is to divide them into functional and non-functional specifications.

The **functional specifications** are:

- The game is controlled through a graphic interface.
- The options that the user has available are: view the stats of the armies (by passing over each planet during a battle) and buying ships, upgrading technology levels and showing reports in between battles.
- The battle develops automatically.
- The game allows the planet to lose up to three battles (when this number is reached the game is over).

The **non-functional specifications** are:

- The game is to be executed in a java environment.
- A new enemy threat arrives to attack the player's planet every three minutes.
- Once the threat has arrived, a battle starts within sixty seconds.
- The data that is generated from the moment a new game is initialized until it is either terminated or finished is safely stored in a SQL database.
- The tables that conform the database store: the planet's name (chosen by the user), the number of units the planet has left (which is updated every after battle), the planet and enemy's armies in every battle and the line-by-line development of each battle.
- The graphic interface's only available language is English.

○ Game Structure (Screens, States, Scenes)

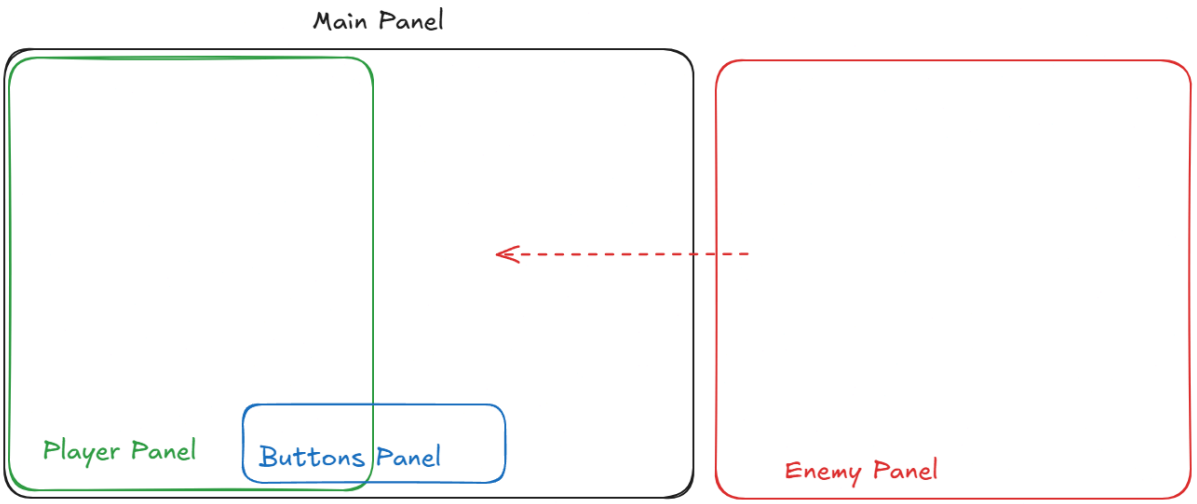
GUI

The Graphical User Interface contains the following windows: main, buy, upgrade and reports.
Each one has its own structure.

Main Window

This window contains four panels: a main panel that draws the background of the universe; a panel for the user's planet; another one for the enemy; and one for the options panel. These panels are all layered over the main panel (the selector and bullet panels, that are visible during a battle, are also within the main panel).

Using a null layout to allow maximum flexibility in positioning, we can place the enemy panel initially outside the visible area, and gradually move it into its battle position. This approach also allows us to move the selector and bullet panels off-screen when necessary.



Inside the options panel, there are three buttons: Buy Ships, Upgrade Technology, and View Reports. Each of these buttons opens a new window where the corresponding functionality is executed.
In the case of Reports, an additional window opens when a report is selected, allowing us to view its content.

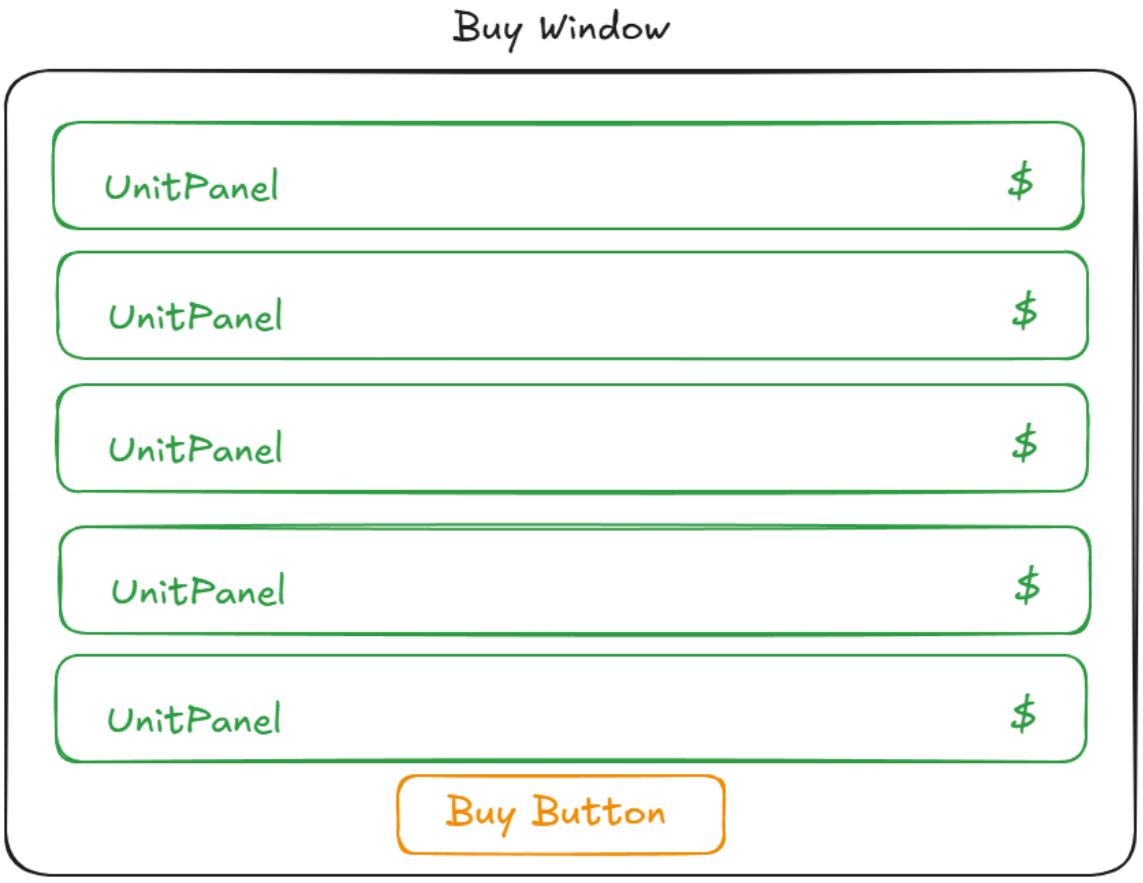
Buy Window

The Buy Window has a main panel that is situated as the background over which every unit panel is situated.

Using a BoxLayout with a y axis parameter to position its components vertically from top to bottom. These components are a units panel for each type of ship and a Buy button at the bottom.

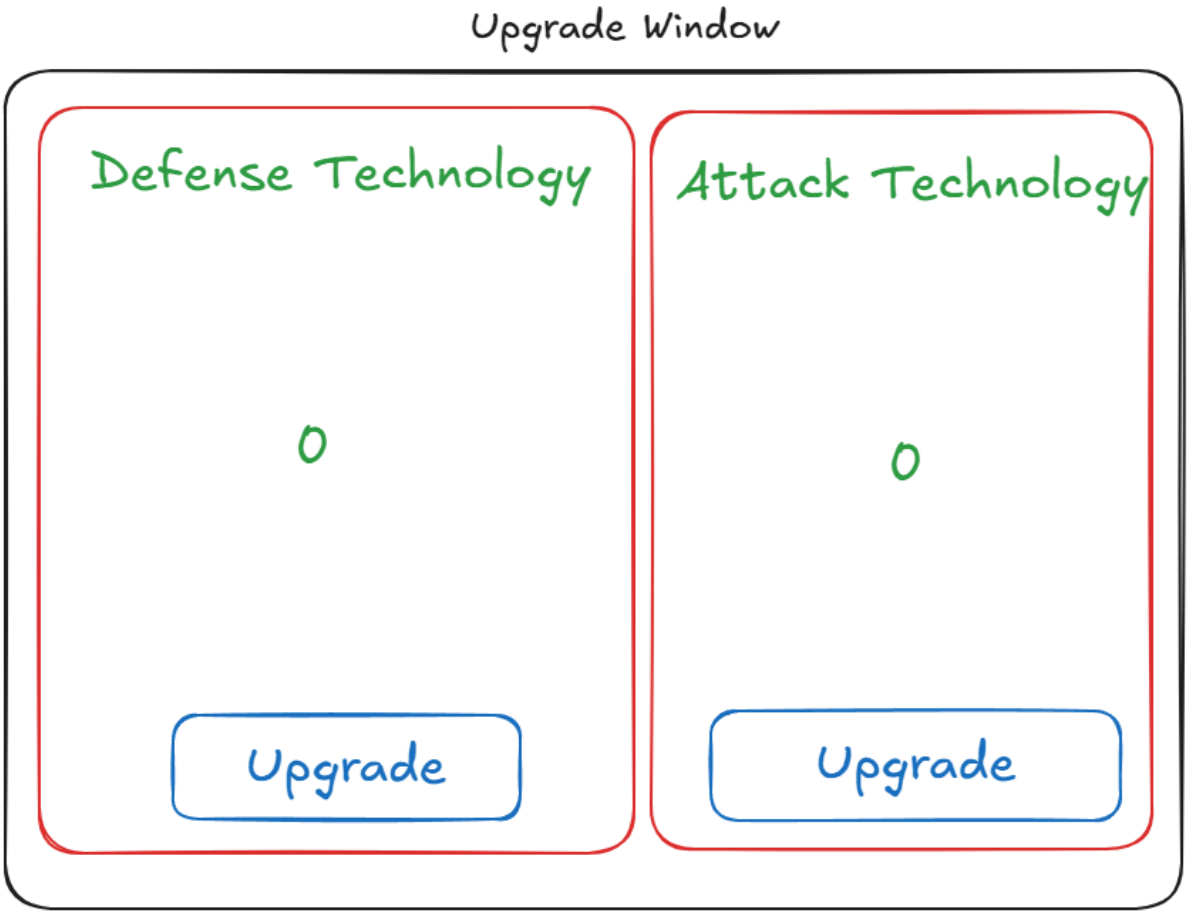
Each unit panel consists of the number of current units that the planet has, an image of the ship, two buttons (situated on the left and right side of the ship's panel which allow the user to choose how many ships he wants to buy), the metal and deuterium cost and the quantity of ships that are currently selected to be bought.

These panels have a BoxLayout with an x axis parameter that positions its components left to right.



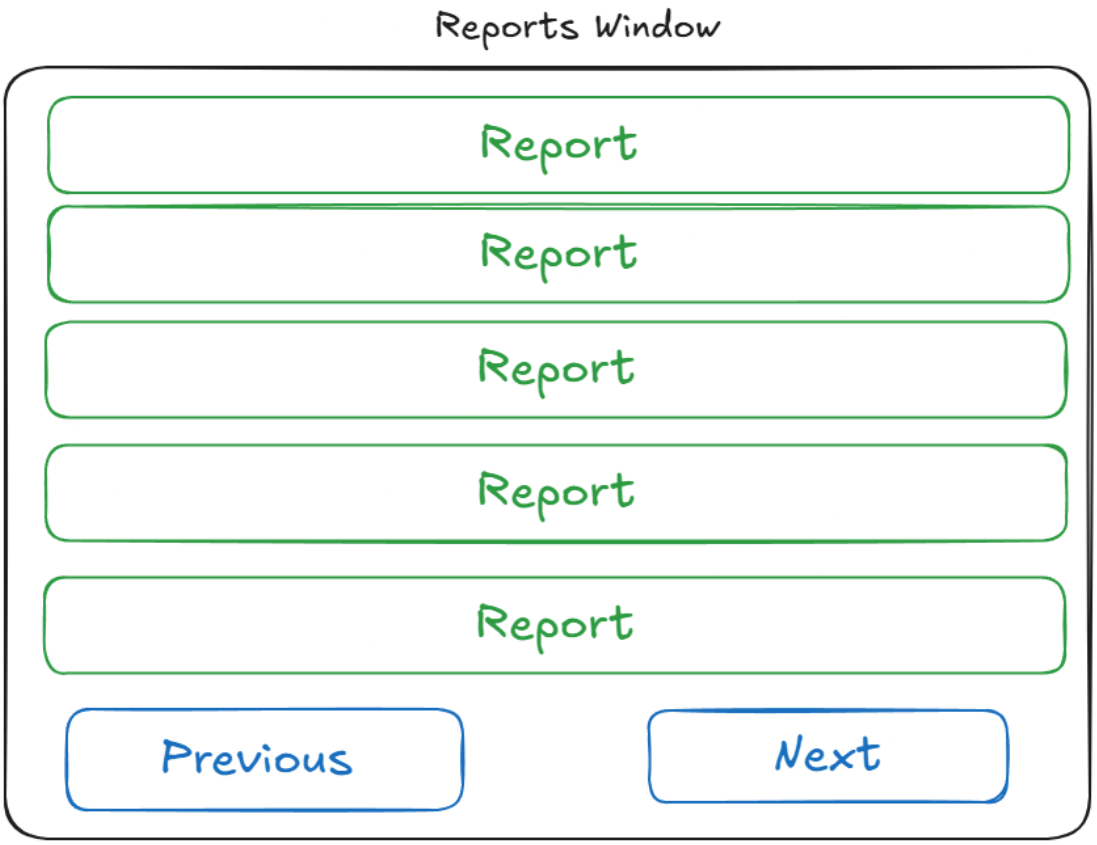
Upgrade Window

This window has a main panel as background that has a BoxLayout that will situate the components from left to right. These components are two panels (one to upgrade the attack technology level and the other to upgrade the defense technology level.
These panels have a main panel with a BorderLayout as a background with a stats panel (that shows the current technology level) that is situated at its center and a button that is positioned on its “south”.
The stats are repainted each time the level is increased.

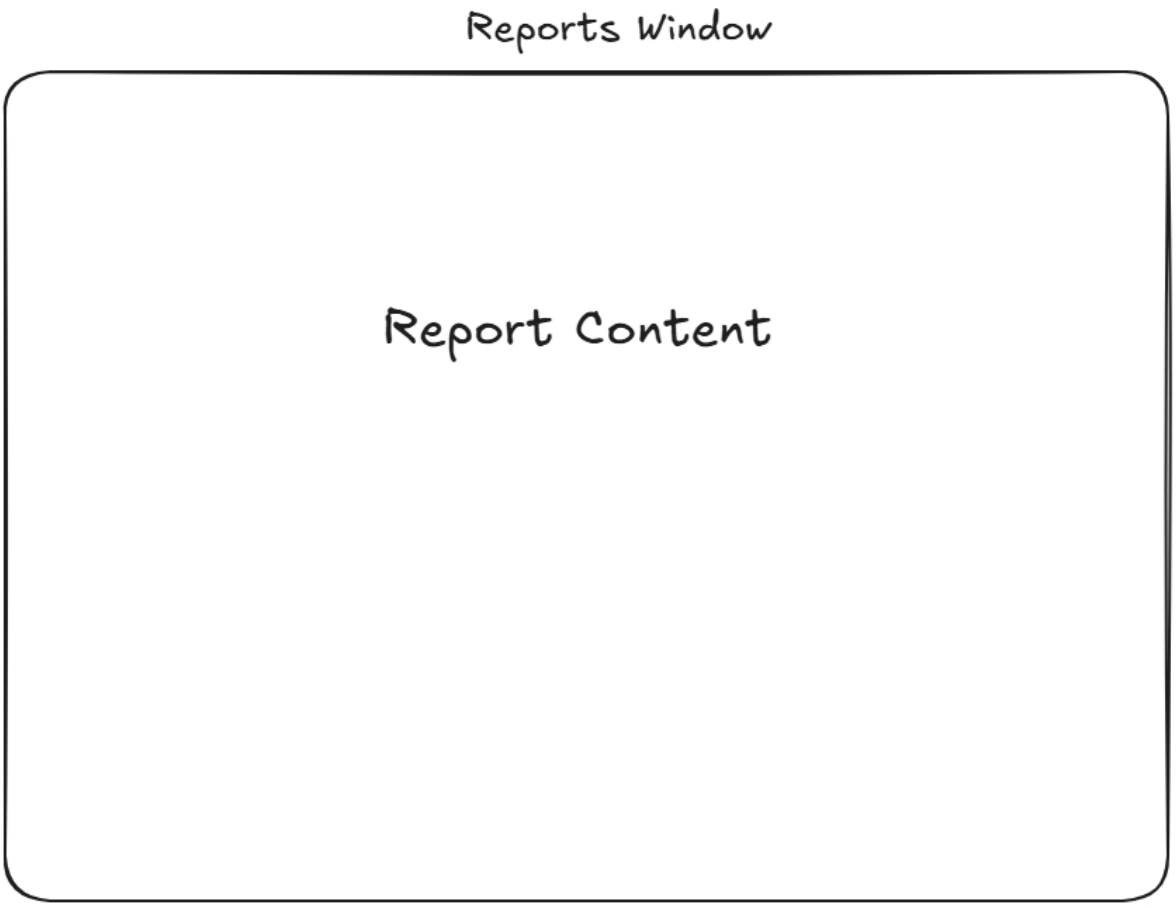


Reports Window

It has a main panel as a background with a BoxLayout with a y axis parameter that positions a data panel and buttons panel from top to bottom.
The data panel has a BoxLayout with a y axis parameter that positions a maximum of ten report panels per page (there is one report for each battle) and a buttons panel with a BoxLayout with an x axis that positions a “previous” and “next” page button from left to right.



If one of the report panels is clicked, a Report Window is opened and the battle summary and battle development are shown.



States

The different game states and the connections between the GUI, the “backend” classes, and the database are managed through controllers. These are classes that hold the necessary references (for example, the InterfaceController contains references to the Main Window, the Planet, the Battle, and also to the Player and Enemy Panel, although these can also be accessed from the Main Window). The DatabaseController holds what is needed to connect to the database. Each controller class includes a static attribute that is assigned to the instance of the class when the component is created. This ensures that the object is properly instantiated and can be accessed from anywhere in the code.

```
public class InterfaceController implements Variables, VariablesWindow {

    private Planet planet;
    private PlayerPanel playerPanel;
    private EnemyPanel enemyPanel;
    private MainWindow mainWindow;
    private Battle battle;
    private String buyStringContext;

    public static InterfaceController instance;

    public InterfaceController() {
        super();

        InterfaceController.instance = this;

        this.mainWindow = new MainWindow();
        this.planet = new Planet();
        this.playerPanel = mainWindow.getPlayerPanel();
        this.enemyPanel = mainWindow.getEnemyPanel();
        this.battle = new Battle();
        this.buyStringContext = "";

        this.mainWindow.approachEnemy();

        planet.generateInitShips();
        playerPanel.setPlayerArmy(planet.getArmy());
    }
```

```java
public class DatabaseController implements Variables{
    public static DatabaseController instance;

    // Connection data
    private String urlDatos = "jdbc:mysql://planet-wars.clmmsosyssic.eu-north-1.rds.amazonaws.com:3306/SpaceWars?
    private String usuario = "admin";
    private String pass = "proyecto2025";
    private Connection conn;

    // Declaring variables that will be used later
    private PreparedStatement ps;
    private Statement stmnt;
    private ResultSet rs;
    private String query;
    private ArrayList<Integer> planet_ids;
    private ArrayList<String> planet_names;
    private ArrayList<ArrayList<Integer>> num_battles;

    public DatabaseController() {
        DatabaseController.instance = this;

        // Connecting to the Database
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");

            conn = DriverManager.getConnection(urlDatos, usuario, pass);
            stmnt = conn.createStatement();

            } catch (ClassNotFoundException e) {
                System.out.println("Error en el driver");
                e.printStackTrace();
            } catch (SQLException e) {
                System.err.println("SQL exception");
                e.printStackTrace();
            }
    }
```

○ Character Design and Gameplay Mechanics

The designs of the different ships have been taken from the following asset pack by the artist Kenney: Space Shooter Extension. To change the ships to the different colors we needed, we used the following script, modifying the output colors each time:

```python
from PIL import Image

def replace_dominant_color(
    input_image, output_image,
    origin_color, final_color,
    tolerance=30
):
    """
        Replaces the pixels where the dominant color coincides with
`origin_color` (RGB) and substitutes them for `final_color` (RGB).
    """
    img = Image.open(input_image).convert("RGBA")
    pixels = img.load()

    for y in range(img.height):
        for x in range(img.width):
            r, g, b, a = pixels[x, y]
```

```
                # Detect whether the pixel is sufficiently close to the
origin color
            if abs(r - origin_color[0]) < tolerance and \
               abs(g - origin_color[1]) < tolerance and \
               abs(b - origin_color[2]) < tolerance:
                pixels[x, y] = (*final_color, a)

    img.save(output_image)
    print(f"Image saved as: {output_image}")


# === USE OF THE SCRIPT ===

# For example: from red (255,0,0) to orange (255,165,0)
original_image = "your_image.png"
result_image = "resultado.png"
origin_color = (255, 0, 0)      # Color to replace (red)
final_color = (255, 165, 0)   # New color (orange)

replace_dominant_color(
    original_image, result_image,
    origin_color, final_color
)
```

The color of the ships is chosen randomly when a new army is created, in the same way as for the planets.The planet assets are also from Kenney (Planets).

The core gameplay mechanic of our video game is based on earning resources through periodic generation by the player's planet and by winning battles. These resources must be used to upgrade technology and purchase stronger ships.
The battle system follows these steps:

1. Check if the player's army has any ships. If not, the battle ends.
2. Randomly select one of the two armies to start attacking.
3. Calculate the attack probabilities for each type of unit.
4. Select a unit type.
5. Randomly choose a unit of the selected type.
6. Highlight the selected attacking unit type in the GUI.
7. Calculate the defense probabilities for each unit type.
8. Select a unit type for defense.
9. Randomly choose a defending unit of that type.
10. Highlight the selected defending unit type in the GUI.
11. Calculate and log the actions taken.
12. Visually execute the attack:
    a. Rotate the bullet panel to the correct angle.
    b. Calculate the bullet's start and end positions.
    c. Compute the bullet's direction and angle.

   d. Rotate the ship and bullet sprites accordingly.
   e. Move the bullet along the indicated trajectory.
   f. Upon impact, hide the bullet and spawn an explosion.
   g. Hide the explosion after a short delay.
13. Check if the defender has died (i.e., their shield is less than or equal to 0).
   a. Calculate whether debris is generated. If so, add it to the current battle's debris list.
   b. Update the defending army's losses.
   c. Recalculate the unit distributions.
   d. Add a battle log entry.
   e. Update the percentage of remaining defender units.
14. Check if the defending army has less than 20% remaining units; if so, end the attacker's turn.
15. Check if the attacker can attack again. If so, return to step 7.
16. Check each army's remaining percentages. If either is below 20%, the battle ends.
17. Hide the selectors.
18. Determine the winner of the battle and update the database.
19. If the player has lost 3 times, the game ends.
20. Generate an HTML report of the battle.

○ UML Diagrams (Class Diagrams, Sequence Diagrams if applicable)

Link to UML diagram

3. Architecture and Development

○ Description of the source code structure.

The classes that conform the game are divided in packages with descriptive names, such as planets, ships, battle, gui and so on. To see a visual representation of the file system, view the file diagram in its specific section.
The classes functionalities are divided into methods that are called to in the constructors. There is a series of variables that are stored as constants in interfaces for the whole project to use.
These classes and functions are integrated in controllers that facilitate the "communication" between classes.

○ Explanation of the main components: Player, Enemies, Projectiles…

The main “protagonists” of this game are the player and the enemy.

The player gets to choose his planet’s name at the beginning, before the creation of the player’s planet.
The planet is created with an initial amount of metal and deuterium and a level zero for both attack and defense technologies. Its initial army is initiated with a ship of each type (a number that can be increased by buying ships with the available resources).

The enemy is regenerated for each battle. This enemy has a certain quantity of resources that are calculated as a fraction of the initial resources the planet has and a slight increment in the enemy’s resources for each battle that has occurred. From these resources, the enemy’s army is randomly formed. While the enemy has enough resources to buy at least a light hunter (the less expensive ship), a random number will be generated and, depending on the number, the program will check whether the enemy has sufficient resources to afford the specific randomly-chosen ship. And if so, one of those will be added into the army.

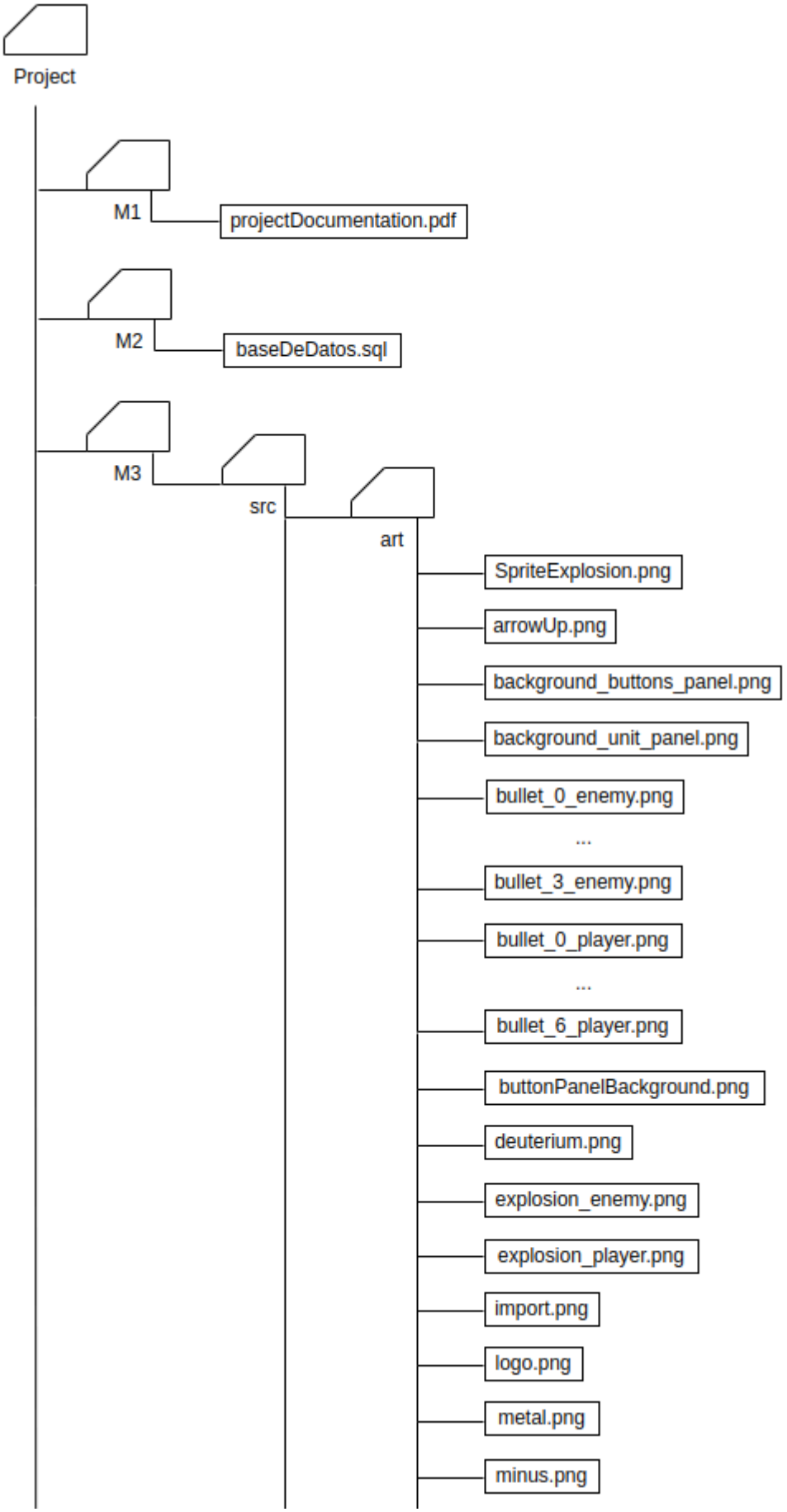
These two protagonists combat each other in a battle. In the battle, it is firstly randomly decided whether the player or the enemy starts attacking. Once it is decided, the type of ship that attacks is chosen from the ones the army has available. Then, a ship is chosen from the defender’s army. The attack occurs and it is randomly determined if the attacker gets to attack again or the roles get reversed (if the planet was attacking, it will defend and viceversa). The battle will end the moment an army gets to less than 20% its initial size.

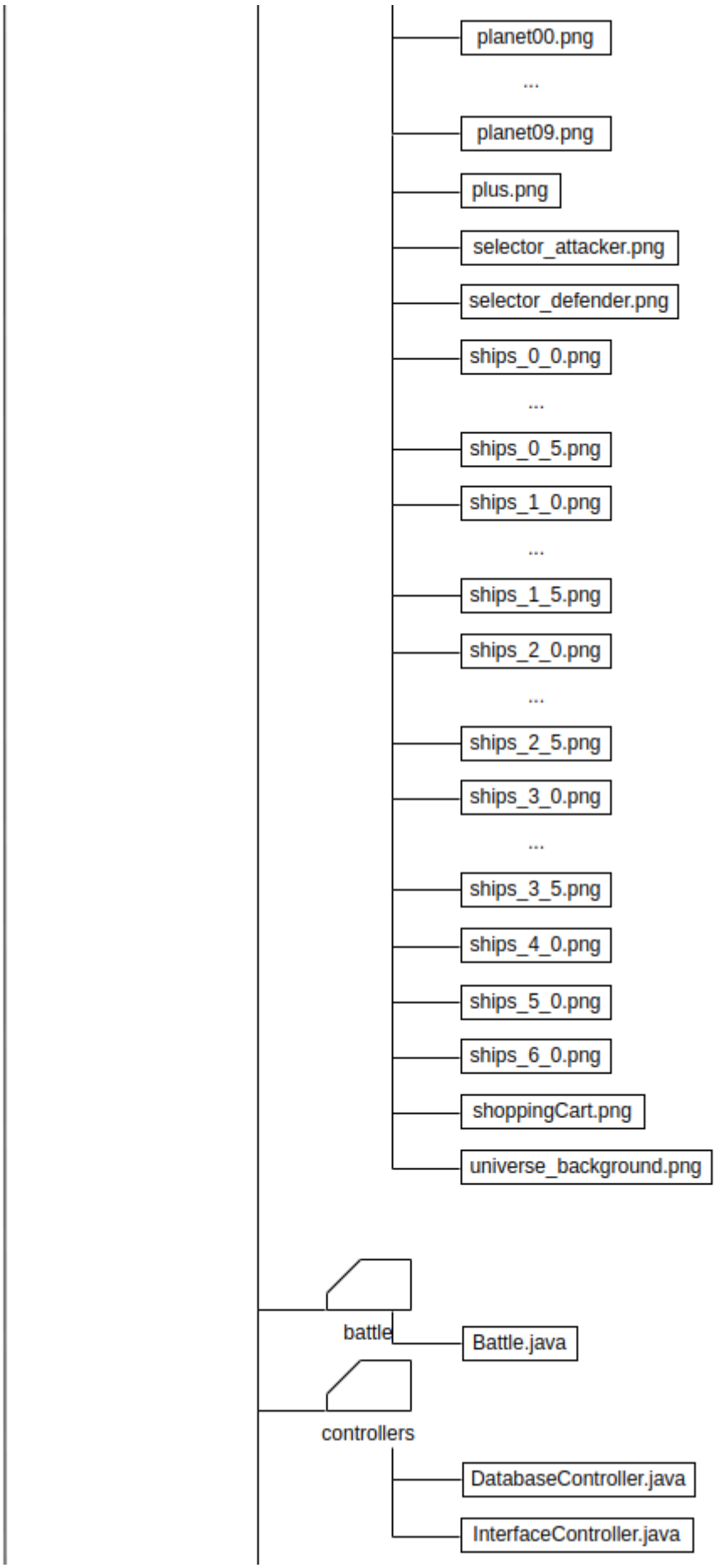
In the graphic interface visualization of the game, half of the screen is dedicated to the planet and the other to the enemy. In these spaces there is an image of each type of ship that the army has (i. e. if the planet has three battleships, a missile launcher and a plasma cannon, in the planet’s half of the screen there will be an image of a battleship, a missile launcher and a plasma cannon). When the attacker’s and defender’s ships have been determined, these are “highlighted” by having an arrowhead pointing downwards above these two ships. The attacker rotates to face the defender and a projectile appears and goes from the attacker to the defender. Once the projectile has reached its target’s position, an explosion image appears to represent the explosion caused by a projectile colliding with a ship. If the ship is destroyed and it was the last of its kind, the image of this type of ship turns invisible.
If the mouse passes over any of the planets (either the player’s or the enemy’s), the stats of this participant will be shown (ship quantity and technology levels and resources in the case of the player).

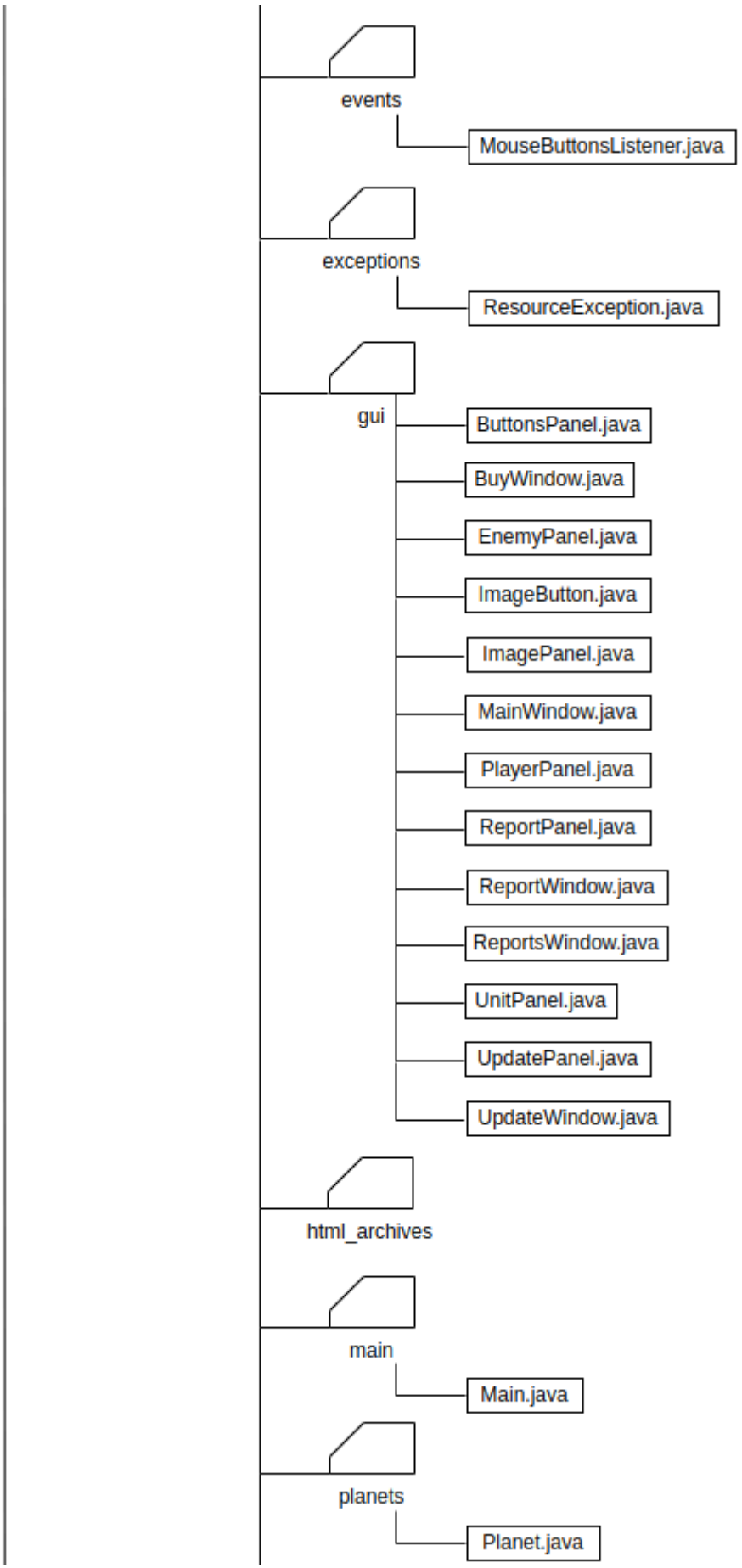
○ File/Project diagram.

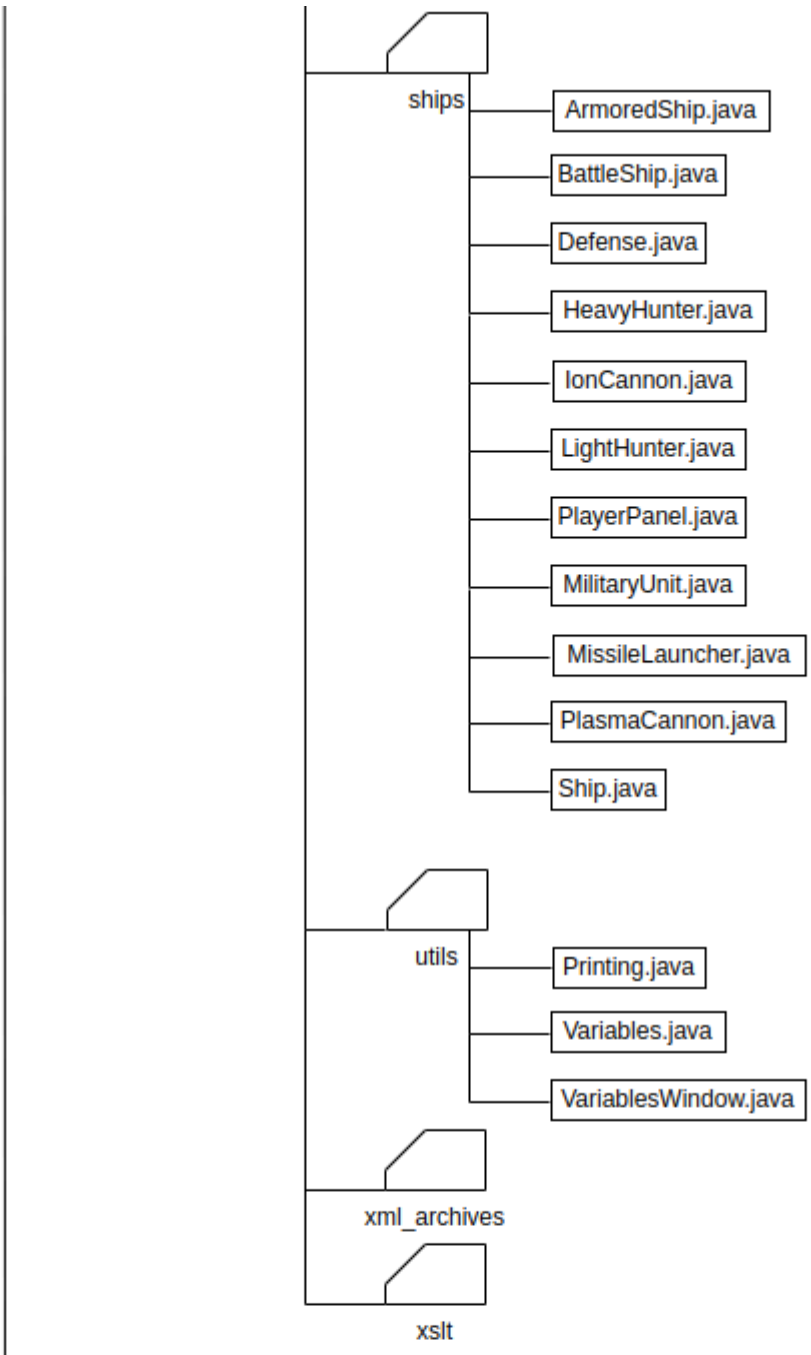
The structure of the files for the whole project is as shown below (the complete repository is represented, not only the game’s code):

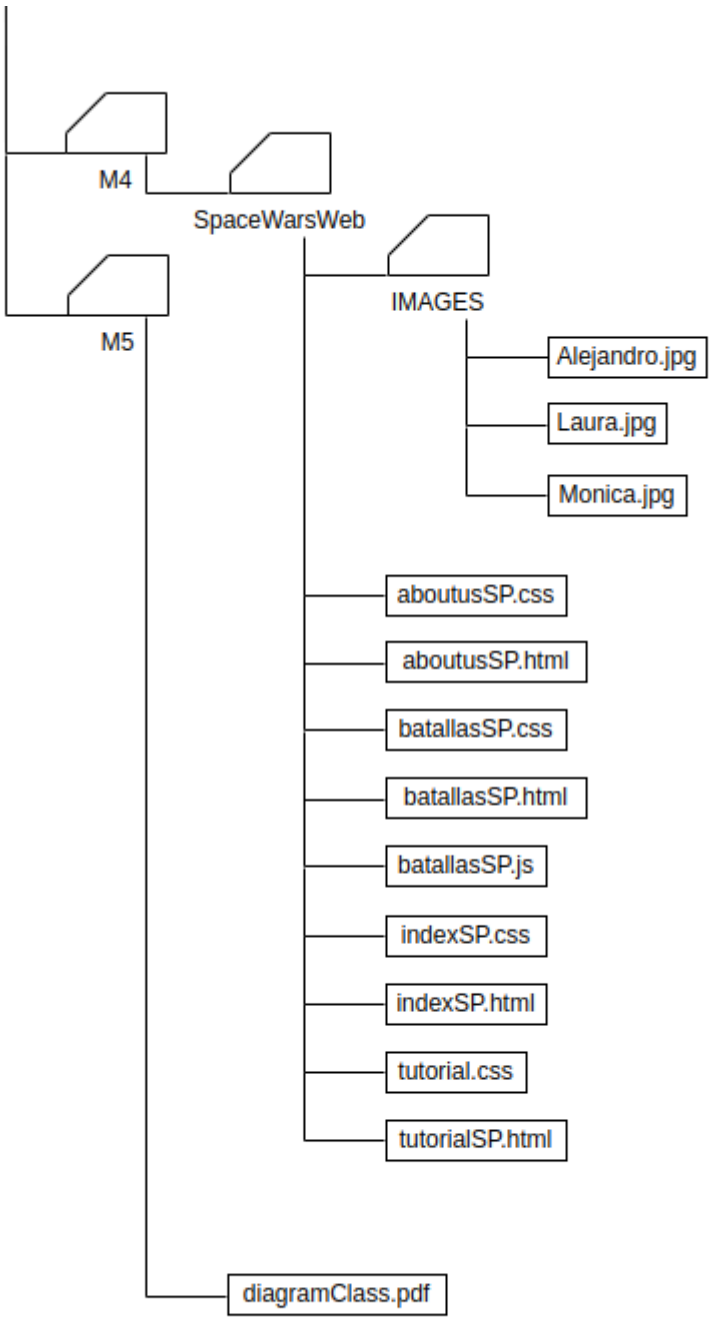
Mónica Domínguez
Laura Toro
Alejandro López




planet00.png
...
planet09.png
plus.png
selector_attacker.png
selector_defender.png
ships_0_0.png
...
ships_0_5.png
ships_1_0.png
...
ships_1_5.png
ships_2_0.png
...
ships_2_5.png
ships_3_0.png
...
ships_3_5.png
ships_4_0.png
ships_5_0.png
ships_6_0.png
shoppingCart.png
universe_background.png
battle
Battle.java
controllers
DatabaseController.java
InterfaceController.java


events
MouseButtonsListener.java
exceptions
ResourceException.java
gui
ButtonsPanel.java
BuyWindow.java
EnemyPanel.java
ImageButton.java
ImagePanel.java
MainWindow.java
PlayerPanel.java
ReportPanel.java
ReportWindow.java
ReportsWindow.java
UnitPanel.java
UpdatePanel.java
UpdateWindow.java
html_archives
main
Main.java
planets
Planet.java


ships
ArmoredShip.java
BattleShip.java
Defense.java
HeavyHunter.java
IonCannon.java
LightHunter.java
PlayerPanel.java
MilitaryUnit.java
MissileLauncher.java
PlasmaCannon.java
Ship.java
utils
Printing.java
Variables.java
VariablesWindow.java
xml_archives
xslt



○ Data flow and game events.

The execution of the program implies a series of database operations.

1. When the program is started, a new planet is created and, therefore, inserted into *planet_stats*.
2. The *battles_counter* on *planet_stats* gets updated by one.
3. When a battle happens and is completed, a new *battle_stats* entry is created (the method used to insert this returns us the battle id (*num_battle*).

4. The remaining alive units of the planet's army are updated in the *planet_stats* table.
5. The initial and destroyed units of each type of ship are uploaded for the planet (into *planet_battle_army* and *planet_battle_defense*) and for the enemy (*enemy_army*).
6. The battle development is uploaded line by line into *battle_log*.
7. The summary report is generated and shown. To generate this report, there is a series of queries that need to be executed to gather the necessary data.

The events in this game, the actions that get triggered when you interact in a certain way with the graphical interface, can be divided into the mouse over events and the clicked events.

Mouse Over Events

In the waiting time before a battle, if the mouse is over any of the three buttons at the bottom of the screen, the buttons will go through a subtle darkening in color.
If the buying button is clicked, in the buying window, the plus and minus buttons have the same darkening when the mouse is over them.

During a battle, if the mouse is over one of the planets, the stats will be shown. Or if it is above one of the ships, the name of the ship will appear along with the number of units that the planet has that are still alive.

Clicked Events

On the other hand, the vast majority of the events happen when a button or panel is clicked. In the main window, the click of the buy button will open a buying window in which the plus button highers the number of units to buy, the minus one lowers it and the buy one at the bottom executes the buying.

If the upgrade button is clicked, an upgrade window is opened and there are two buttons in there that allow the upgrade of the attack or defense technology levels.

If the show reports is clicked, a reports window is opened and in there if the panel of a battle is clicked, its report will be shown. Also, there are two buttons at the bottom (previous and next) which allow you to navigate through the available battles to choose from.

4. Test and Debug

○ Tests performed (unit or manual)

The tests were performed manually, by playing the game and using print statements. Various manual tests were carried out during the development of the game to check that each part of code that was being written worked properly.

Some example of these are: printing the cost of each ship to test whether those were correct, manually adding the cost of multiple weapons to see those totals were being

correctly calculated, testing the probabilities of being chosen of the different types of ships, printing and testing the queries in the workbench to see if these were doing what we wanted them to do…

○ Errors found and how they were resolved

1. **Error when sending reports to the database on the school computers.** The school computers were not able to send data to the database within the short time available for testing. The testing period was extended, and the issue no longer occurs.
2. **Errors with the database queries.** The names of some columns were changed later in the development process which caused methods that previously worked to throw exceptions, these were easily resolved by updating the column names in the queries, once we realised the issues' cause.
3. **Having planet and battle ids combinations that did not exist in the database.** The battle id was being initialized as 0 every time a new battle began. Therefore, making it impossible to accumulate the number of battles that happened in a game, which we realized later on and fixed by erasing that initialization.
4. **Errors being thrown when the planet army started a battle with zero units.** This was a situation that we hadn't originally thought of. When we tested it, we realized that if we started the battle without checking whether the planet army units were above zero, there would be a division by zero. This was fixed by making this check before starting the battle.

○ Improvements implemented during development

1. We ask for the player's name when starting the game.
2. When the player loses 3 times, the game ends.
3. You can get the summary and development reports for every battle and not just the last five.

5. Multimedia Resources

○ Fonts used (with usage rights if applicable)

- DejaVu Sans Mono: when a text is painted in a panel this font is used.
- Monospaced: when a string is shown in a JTextArea (as it is done when a report is being viewed), to properly see the format that has been implemented on the string, this type of font is required.

○ How they have been integrated into the project

Both fonts are installed and incorporated with Java Swing, which is imported to build the graphic interface.

6. User Manual

○ Instructions to execute the game

1. Download the project from releases.
2. Get Java JDK if you do not have it installed (JDK Download).
3. Open Eclipse IDE (Installation Guide on Windows | Installation Guide on Linux).
4. Open the project clicking on **File** > **Open Projects from File System...** and select the M3 folder you have downloaded.
5. Download MySQL Connector library.
6. Right click on your project and select **Build Path** > **Configure Build Path...**.
7. Go to Libraries tab and change *mysql-connector-8.0.15.jar* for the file you have just downloaded. In case you do not have it, you must click on **ModulePath**, then click on **Add External JARs..**. and select the file.

○ Game controls and objective

Game Controls

You will have to use your mouse to point and click in different parts of the graphical interface. No keyboard or gamepad are needed.

Using your mouse, for example, you can click the buttons in the menu at the bottom of the screen to buy ships, upgrade technology or show reports. Also, if you position your mouse above either one of the planets on screen during a battle, the stats will be chosen.

Objective

Survive through enemy waves that threaten your planet and try to conquer it. If you fail 3 times, your planet will be conquered and you will lose. To prevent it, you can improve your technology level (which will increase the damage of your attacks and the resistance of your armors) and buy ships (to substitute those that have been lost in battles).

7. Reflection and improvements

○ Found difficulties and how they have been overcome.

Whether the logic of the game was somewhat similar or comparable in difficulty to other programs or games that we have coded, incorporating a graphic interface to the mix was definitely a challenge.
This difficulty was overcome by having controller classes that were able to reference key classes and, for example, allow methods of another class to be used without an explicit instance in the class that it was needed in (avoiding having to construct them again).

Another issue we faced was to realize the source of apparently random errors that occurred. Errors that happened in some laptops and not others, methods that connected to the database and failed in different places without apparent reason (with console errors that were confusing).
Though, once we realized that it was a matter of variables being overwritten by following battles when the current battle hadn't been completely updated into the database, the solution was simple. By increasing waiting times, these errors were overcome.

Also, having to face the rotation of images and other visual effects that made the game more realistic was another problem that was overcome by research and trial and error.

○ Future improvements or new functionalities.

Every project can be improved and further functionalities can always be implemented with longer development available time. Some of our proposals of improvement are:

- Being able to choose the level of threat you want the enemy to pose: low threat (the enemy has fewer ships and with less attack and defense capacity), medium threat (the enemy's army has a proportionate amount of ships with the original attack and defense capacities of each type), high threat (more ships and with more attack and defense).
- Having lower chance of attack and lower chance of attack again when you lose a ship.
- Being able to graphically choose which ship attacks and which ship defends.
- Showing with a JProgressBar the remaining armor of each ship.
- Implementing different types of ships (with different attack and defense values) depending on the enemy planet that appears to threaten the player's planet.