Proyecto Final - PLANET WARS

DAM Curso 2024-2025

Alumnos:

Álvaro Armas Jurado (MIX1)

Rubén Bellido Navarro (MIX1)

Aleix Linares Sousa (MIX1**)**

Index

1. Introduction

1. Game description

Planet Wars is a turn-based game in which you build your space army to win battles against your enemies.

You start with a few units in your army and some resources which grow over time that are used to build new spaceships and improve your power. With your army you engage in battles against enemy armies, and the player who loses more resources in the fight loses it; the winner gets to win some additional resources to improve his army even further.

We have developed this game for the first year final project of our vocational studies degree in Multiplatform Application Development.

2. Project goals

We were given some documentation about the project with the minimum requirements and two weeks to finish it.

Our main goal during the project was:

- Fulfill the minimum requirements of the given documentation
- Achieve a functional and aesthetical result in our game
- Expand it with features that we found interesting

3. Tools used

These are the tools we have used in this project:

- For the development of the game we used the Java programming language and Swing library to make the Graphic User Interface.
- For the website we used the stack of HTML/CSS/JavaScript.
- To share our work and control the progress of the code we used Git. And the work was stored in the online repository GitHub.
- For communication between teammates we used Discord, and for scheduling the work we used Trello.

2. Analysis and Design

1. Functional and non functional specification

This is the minimum criteria that our game should be able to do focusing the functionality:

- Being able to create Planet and new units for his army.
- Show all the information to the user in any moment through the use of a GUI.
- Run battles with enemies that follows the rules given in the documentation.
- Being able to buy new stuff with our resources, that grow both over time and by winning battles.
- Storing all the information in an online Oracle database, and modify the data when needed.

The criteria that our game achieves in a non-functional way is the following:

- The game looks good and is visually pleasant to play.
- It responds well no matter what. Users cannot break it with any combination of actions.
- The code is easy to understand and modify as it is intuitively built and well documented with comments and diagrams.

2. Game structure

The way that we decided to structure our game since the beginning of the project has stayed the same during the whole process as we found a good solution early in the project, and most of it was given to us in the given documentation.

We have a JFrame in the class MainScreen in which we will put everything we need to interact with the program. In our window we will have different panels, each with his own class, that will control and display different parts of the game: shop, game settings, our planet information, and the graphics area.

Our graphics area consists of two possible modes:

- We are either out of a fight, in which case we can spend our resources expanding our army, changing the game settings, etc.
- We are in a fight. In which case, we can see how the fight develops in turns and our graphics area turns into a display of the combat.

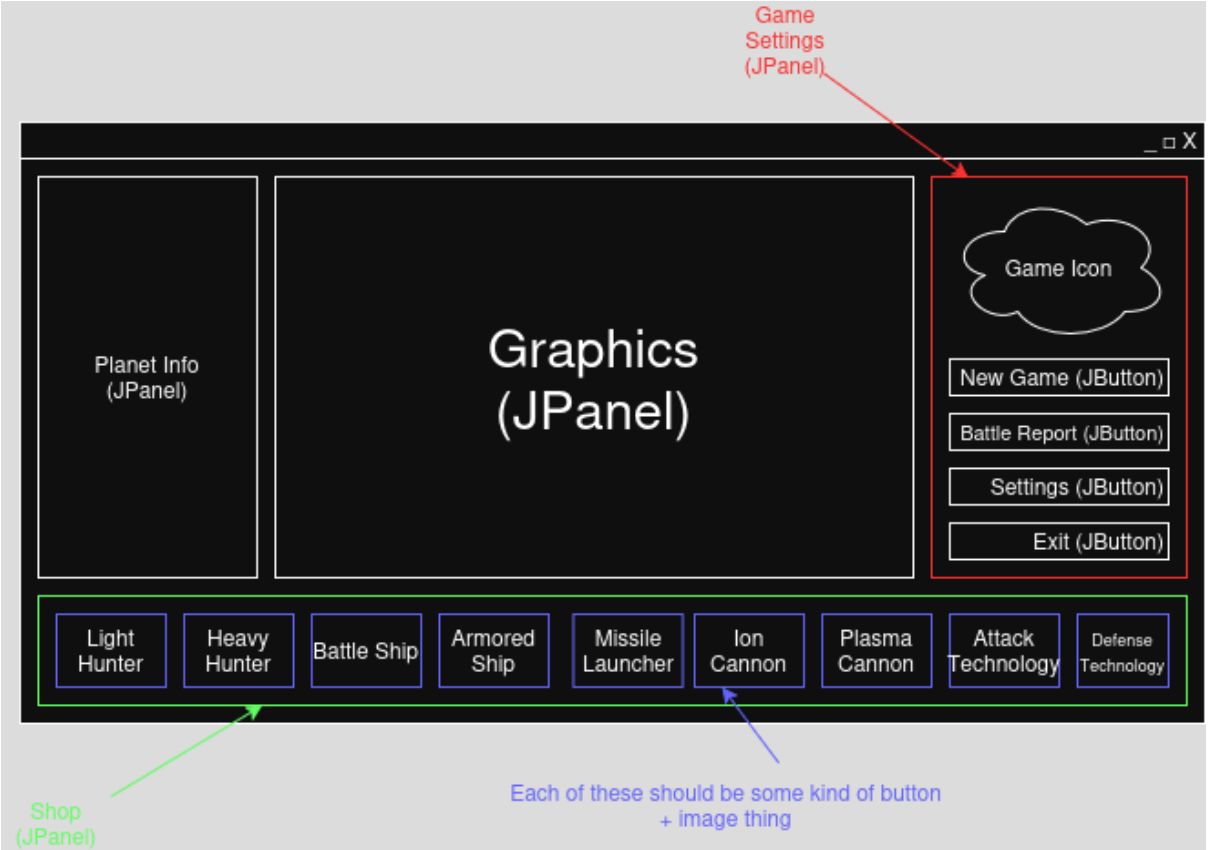
To make possible the proper function of the GUI, we have the logic of our program running under it; that's the use of objects of the classes we defined earlier (Planet, Battle, all the Database objects, etc.).

3. Game mechanics

We decided that our game would be based on decision-making and no controller mechanics would be implemented in it. So in order to play, you are only asked to select buttons on the screen which will give you access to all the possible options within the game; and those are buying things and initializing battles with enemies.

4. GUI design

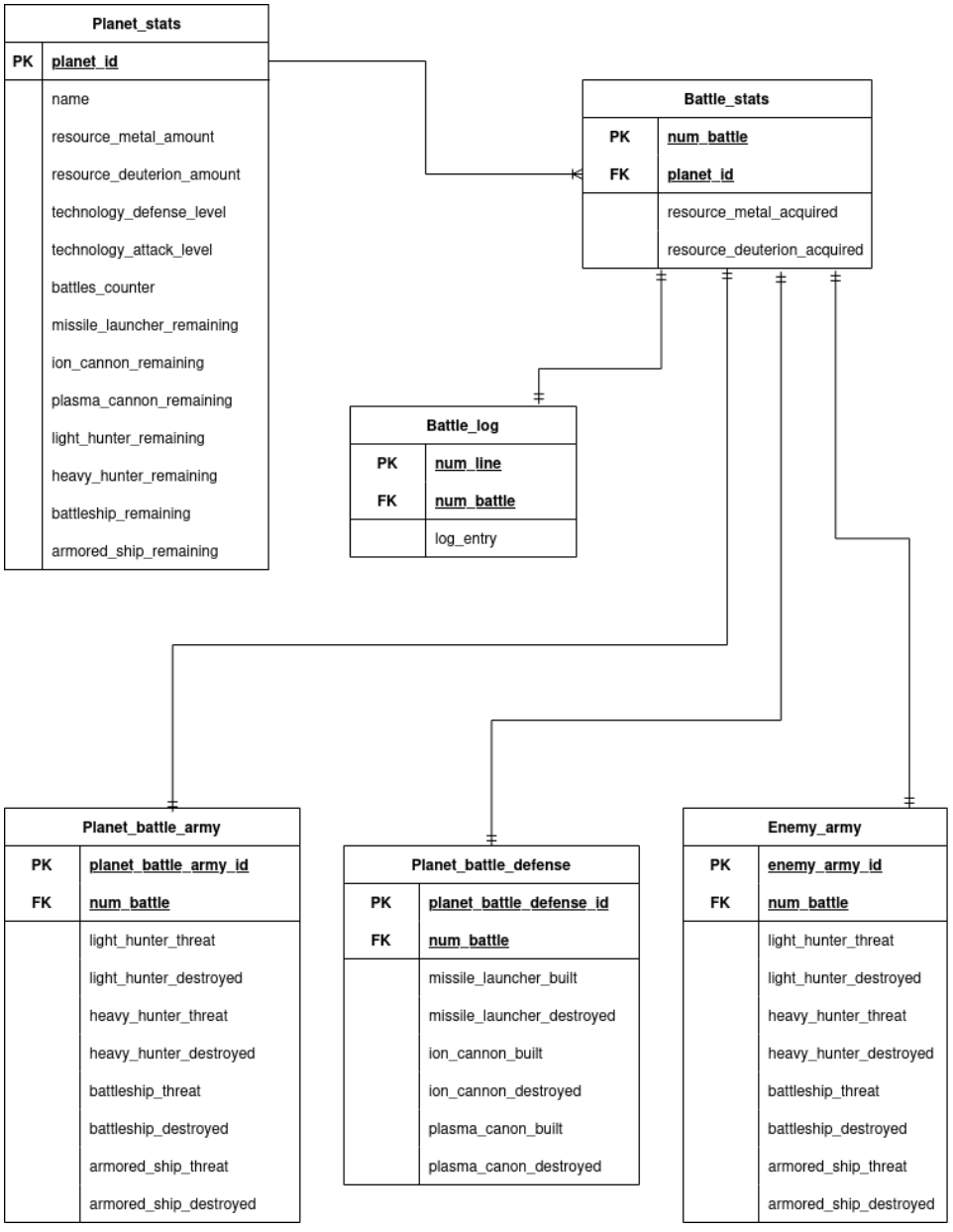
To define how the game should look like in our GUI, we did a sketch in draw.io:



We have mostly stayed with this original design, but some decisions were made based on our criteria while trying new things as can be seen in the end result shown in some of the screenshots of this documentation.

5. DDBB diagram

We followed a similar database structure to the one that was given to us in the given documentation, but changed some of the keys in it to fit our needs. Here is the diagram:



6. UML diagrams

As some of our classes are too large to fit it all in a single diagram, we made a few that will help understanding the program in separate parts.

(Diagrams are shown in the next few pages)

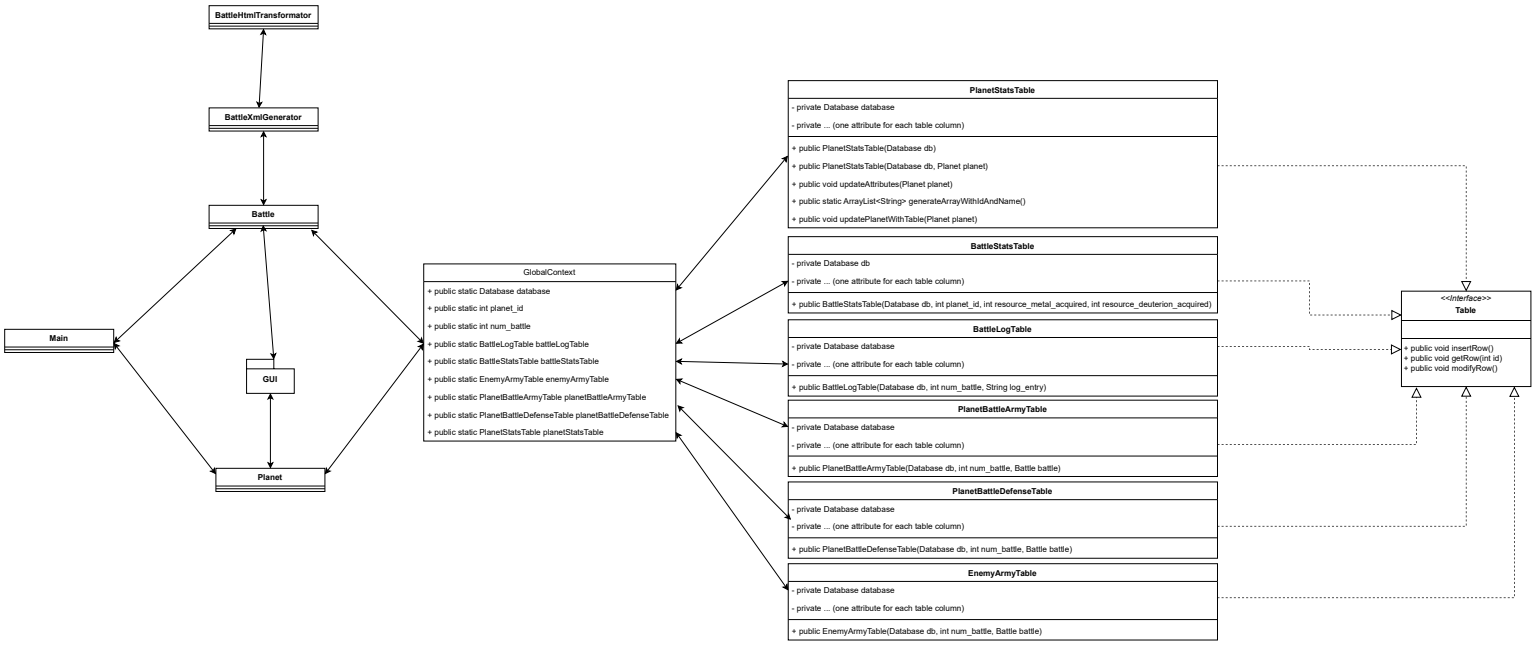
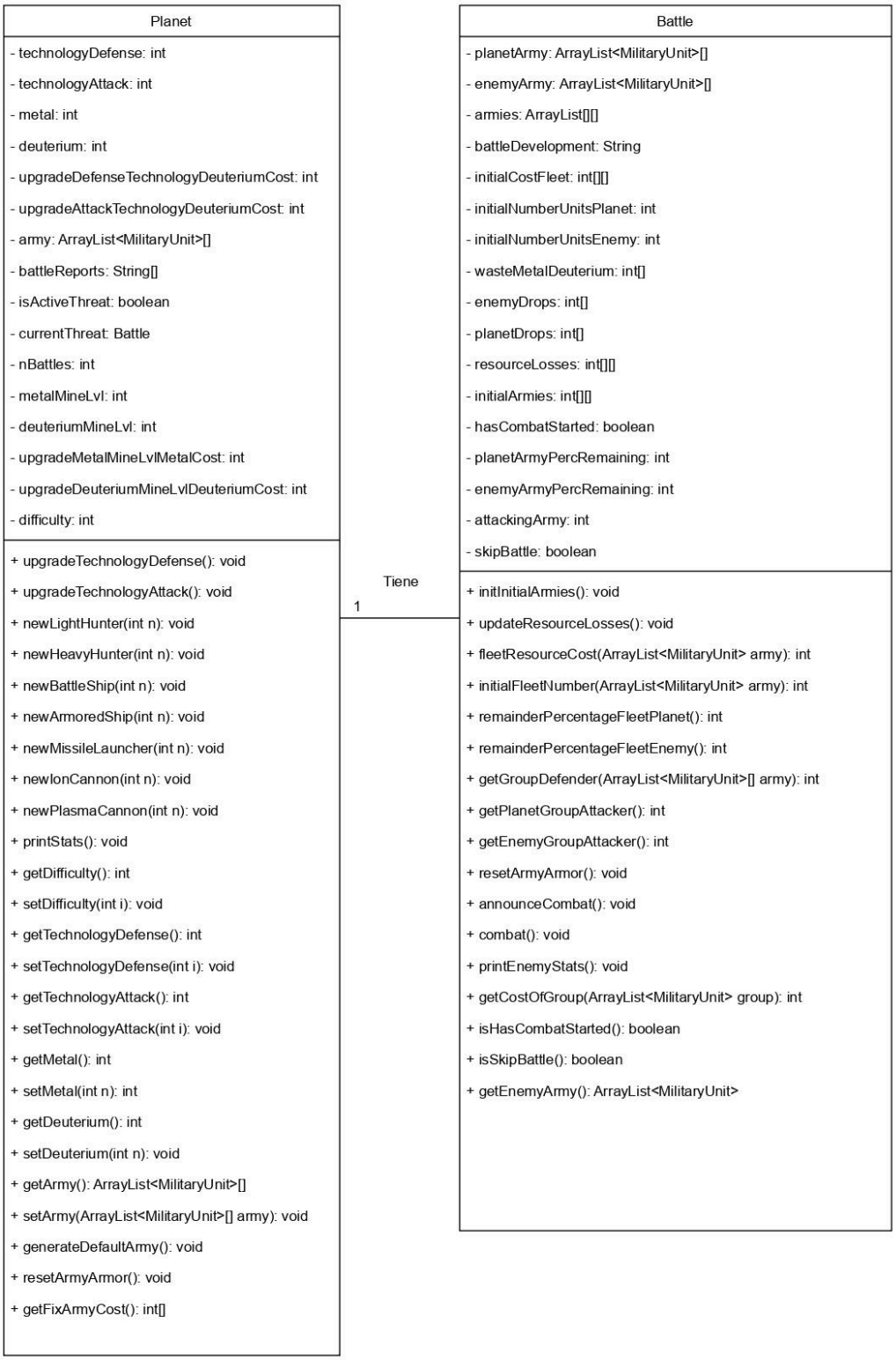
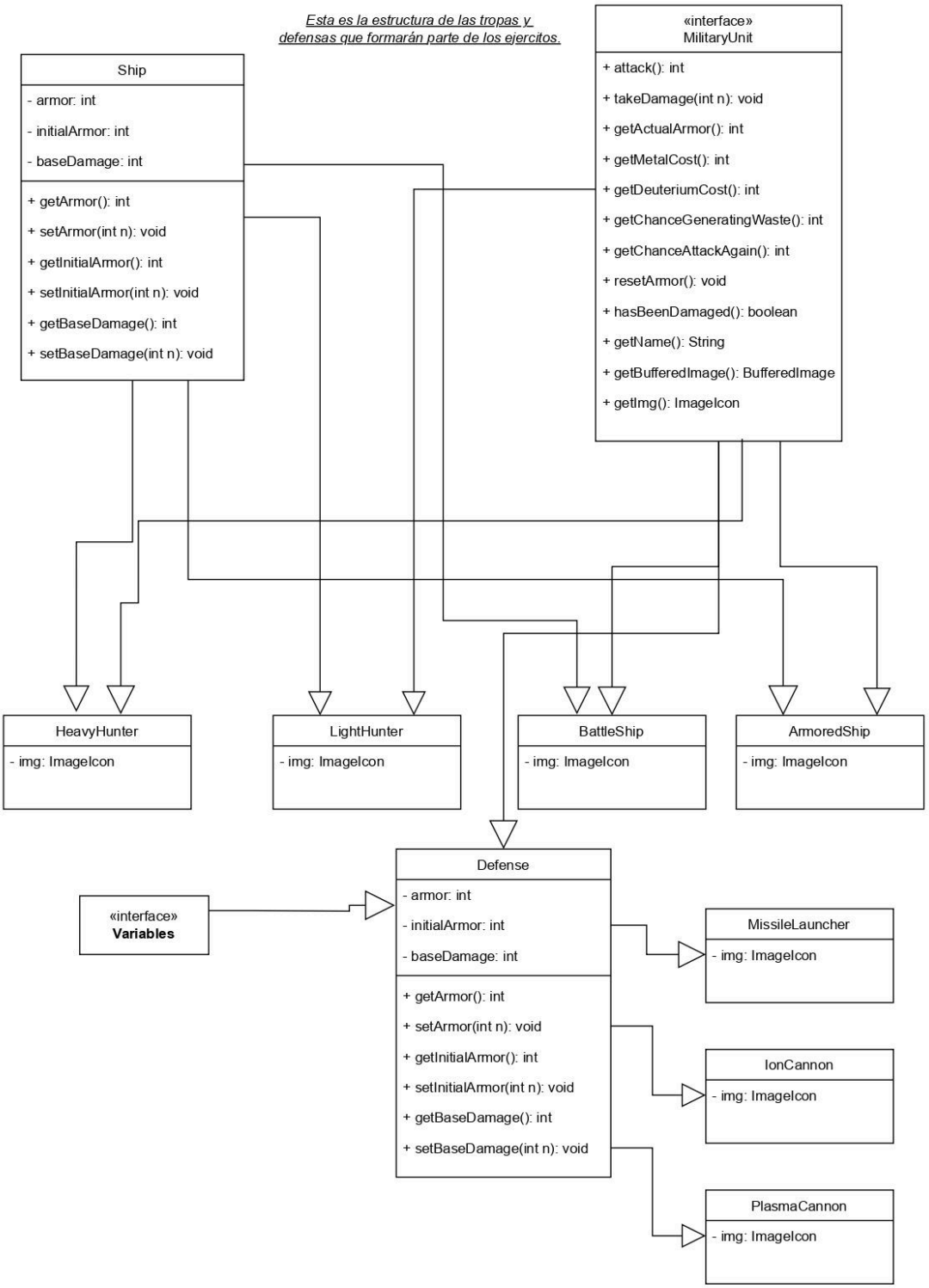

BattleHtmlTransformator
BattleXmlGenerator
Battle
Main
GUI
Planet
GlobalContext
+ public static Database database
+ public static int planet_id
+ public static int num_battle
+ public static BattleLogTable battleLogTable
+ public static BattleStatsTable battleStatsTable
+ public static EnemyArmyTable enemyArmyTable
+ public static PlanetBattleArmyTable planetBattleArmyTable
+ public static PlanetBattleDefenseTable planetBattleDefenseTable
+ public static PlanetStatsTable planetStatsTable
PlanetStatsTable
- private Database database
- private ... (one attribute for each table column)
+ public PlanetStatsTable(Database db)
+ public PlanetStatsTable(Database db, Planet planet)
+ public void updateAttributes(Planet planet)
+ public static ArrayList<String> generateArrayWithIdAndName()
+ public void updatePlanetWithTable(Planet planet)
BattleStatsTable
- private Database db
- private ... (one attribute for each table column)
+ public BattleStatsTable(Database db, int planet_id, int resource_metal_acquired, int resource_deuterion_acquired)
BattleLogTable
- private Database database
- private ... (one attribute for each table column)
+ public BattleLogTable(Database db, int num_battle, String log_entry)
PlanetBattleArmyTable
- private Database database
- private ... (one attribute for each table column)
+ public PlanetBattleArmyTable(Database db, int num_battle, Battle battle)
PlanetBattleDefenseTable
- private Database database
- private ... (one attribute for each table column)
+ public PlanetBattleDefenseTable(Database db, int num_battle, Battle battle)
EnemyArmyTable
- private Database database
- private ... (one attribute for each table column)
+ public EnemyArmyTable(Database db, int num_battle, Battle battle)
<<Interface>>
Table
+ public void insertRow()
+ public void getRow(int id)
+ public void modifyRow()

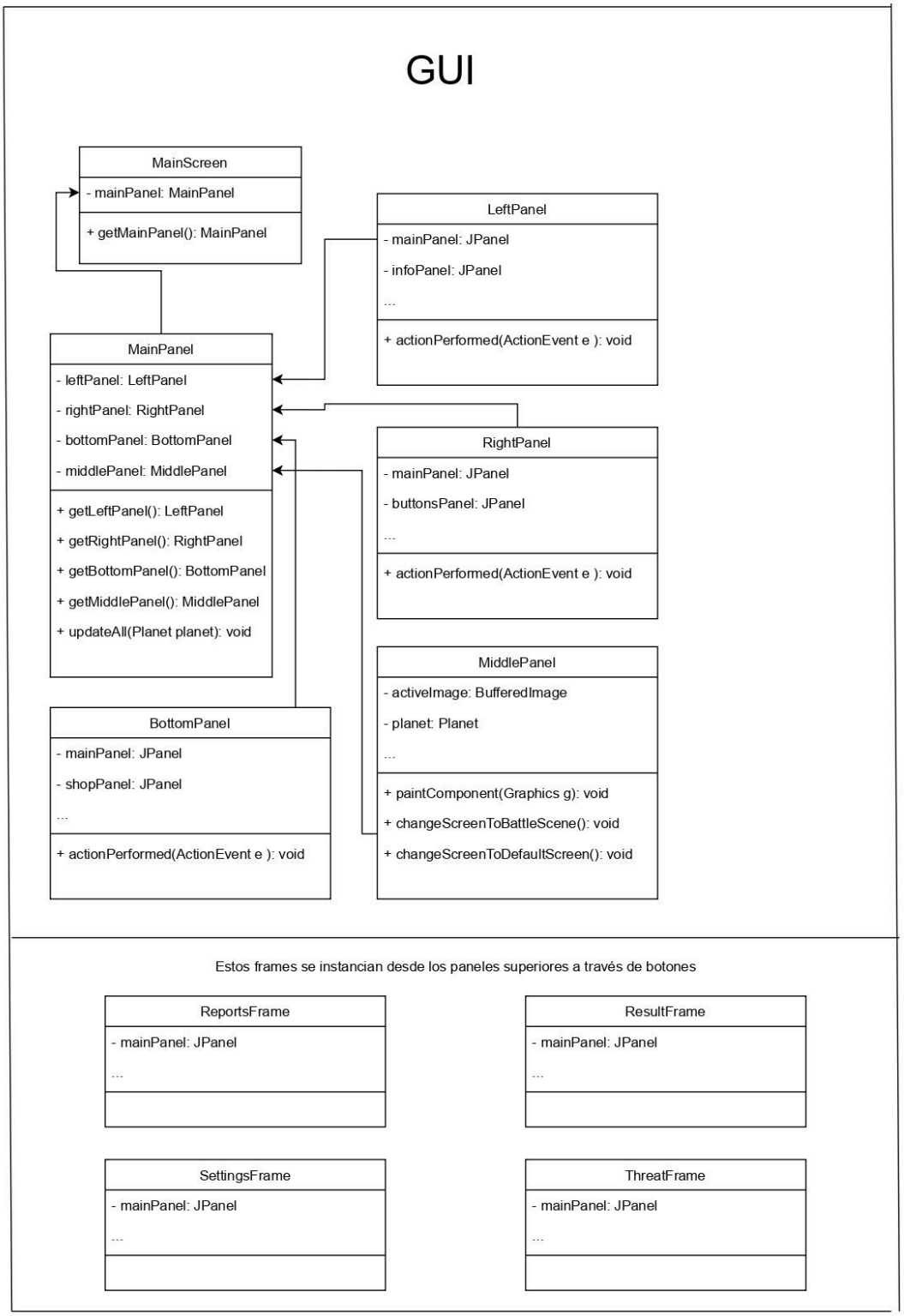
Diagram for the classes Planet and Battle:



All the military units and defenses with their related interfaces:



Classes used in the graphical user interface:



3. Architecture and Development

1. Code structure

The code in our project is divided in 3 packages:

- **classes**: This contains all the classes we used for the logic of the game (Main, Battle, Planet, etc.).
- **GUI**: This package contains all the classes related to the Swing library. They are developed to be responsible only for the graphical part of the game.
- **ddbb**: This contains all the classes used for everything related to our online database. They consist of a class named Database, which is only responsible for making the connection to the actual database; and several other classes that represent each table and are used to modify the content in our database.

2. Main components of the game

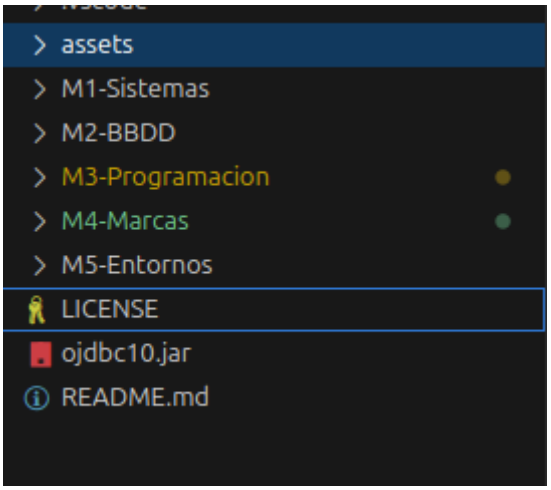
The main components of the game follow a similar structure to the division of classes in packages:

- **The logic of the game,** which controls all the changes that occur inside control variables depending on the actions that happen during the game.
- **The database operations** that allow us to connect to an online database and make changes in it, set new data and retrieve information.
- **The online database**. We chose **Oracle Cloud and his Autonomous Database** as a solution for our project, as it fitted our needs and it's free to use forever unlike other solutions like using a Virtual Machine in Oracle Cloud or other services like Azure or Google Cloud.
- **The graphical part**, which presents all the data to the user and lets him interact with the program.
- **The Variables interface**, which allows us to control different parameters of the game from a single file instead of having multiple variables dispersed in several files.
- A class named **GlobalContext**, which is a class that cannot be instantiated and contains attributes that are both public and static. That allows us to have a class to store all the database related variables and access to them from anywhere in the program with ease.
- Some **support classes** like the ones related to XML and HTML, that allow us to work with those formats during the development of the game.

3. File structure

Our project as a whole is divided in different directories as it was requested in the given documentation. Each directory contains the work that is related to the corresponding subject in our degree.

There are also some files outside these directories: the Java driver that allows the program to make a connection to the database, the README.md and the assets used in that file:



4. Tests and Depuration

1. Testing

For testing the correct operation of the game, we tested it multiple times during the development by playing it. Each time we would introduce a new feature in the game or tweak some of the existing code, we did some testing by trying things in the main method,playing it and looking for any possible unwanted behaviour in the program.

We thought that this approach would be enough as this was a short and not-that-difficult project so we didn't think it was necessary to do unit testing for each element of the program. It would have brought more work to the team than it would have helped us to deliver a better outcome in an acceptable amount of time.

2. Bugs found during development of the game

The biggest problems that we found doing this project were finding behaviours in the program that were not-so-easy to spot right away when we were planning the project. For example, taking into account that when the user is waiting for a battle to start, he could start trying to perform actions that would prevent the correct operation of the program; like pressing New Game, or pressing the Start Battle button several times in a row. But we were able to tackle those as we were discovering them.

There were also some problems related to the logic of our program that were harder to overcome as we ended up with some classes with a lot of attributes and methods; like for example when we decided to implement the mechanic in which our planet can invade other planets, instead of just waiting for being invaded. We could have had an easier time with this kind of problems if we defined classes and methods that are tinier and are responsible for less complex algorithms, as it was hard to quickly analyze our own work at times because of that.

Other than that, there have not been any other problems during the development that were not fixed right away after being spotted.

3. Improvements applied during development

We have applied some upgrades to the game that were not included in the minimum requirements:

- The user can decide if starting a brand new game or loading a previous one when the program starts.
- The player's planet can invade other planets instead of just waiting to be invaded. In that action the player acts as if it was an enemy; so he doesn't have defensive units, while the invaded planet has access to them.
- The player can select the game difficulty by pressing on Game Settings. By doing so some of the variables of the game are tweaked to make combat more challenging or easier.
- The game now features his own soundtrack and sound effects. The song can be muted in the setting in case that the player doesn't want to play it all the time.
- The user can be invaded right away if you don't want to wait by pressing the button "Start a fight".
- The user can have access to any amount of resources he wants in the setting menu in order to test things quickly; that is, we implemented cheats into the game.
- Mine levels were added to the game, which allows you to get more resources over time.
- The user can skip the battle by pressing the space bar in case you don't want to watch the whole fight taking place.
- Users can fix his own army by pressing the button Fix Army. Doing so costs some resources.
- The user can start a new game by pressing the New Game button without needing to close the program in order to open it again.

5. Graphic and Audio Resources

We got most of the artwork from OpenGameArt.org, like the sprites for the starships. The backgrounds were taken from pinterest.com.

The sound effects and song were taken from OpenGameArt.org.

6. User Manual

1. How to play the game

To play the game, you need to have:

- The project is downloaded somewhere on your PC
- The Eclipse IDE installed
- An internet connection

The project can be downloaded from our GitHub repository, and Eclipse can be downloaded from the official website. If you have any problem in the process, the internet is full of tutorials that will help you out.

Once you have those, you need to import to install the project in Eclipse (there are a lot of quick tutorials on that too) and run it from the class Main, which is located in the “classes” package in the M3 directory. That is all you need to play it.

In order to play the game as intended, you should be connected to the internet. In that way, you can store your save in our online database and load previous saves. If you don’t have an internet connection the game will work just fine, but some error messages may show up in the terminal of Eclipse.

2. Game controls and objective

All the game is controlled with your mouse, you just have to click buttons in order to do actions like buying ships or starting a fight. You may use the space bar key if you want to skip the fights.

There is no defined objective, you can play the game until you get bored. You cannot lose the game either; if you are left without an army and have no resources to buy any spaceship, you can just wait until you have generated enough resources to start all over again!

7. Conclusions

After all this work, we think we have done a good job on this project. All the minimum requirements are met, we implemented some new features on our own and also achieved an aesthetically pleasant result in our game. We also managed to finish the project working with consistency in these two weeks so we didn't have to do any kind of last-minute overtime to get it ready for delivery.

Some things can be improved too. We could have divided the work better and plan the tasks more efficiently. We had some punctual problems when merging some of our branches in Git because we didn't know that each other was doing changes too in the same classes, which led to some confusion. And as we said previously, we ended with some huge classes that could have been better planned in order to make things easier to analyze and tweak.

We also had this idea of implementing some 3D functionality in the game using only Swing. One of the members of the team was working on that on his own and we thought it would be interesting to implement it somehow; but the team decided that it was too out of the scope of the project and it didn't contribute enough to the project to justify the extra work.

In conclusion, we are satisfied with the work we did in this project. We learnt a lot in these two weeks not only about technical work but also about teamwork.