# PROJECT DOCUMENTATION

## Planet Wars

Mohamed Adghir
David Perera
Daniel Pons
19/05/2025

# Introduction

Planet Wars is a simulation and strategy game based on the popular browser game OGame. The player manages a planet that must defend itself from enemy attacks by building a fleet of ships and defenses, improving technologies, and managing resources (metal and deuterium). Battles are resolved automatically following turn-based mechanics, where military units (ships and defenses) face off according to their predefined statistics and probabilities.

## Objectives

The main objectives are:

- Implement the game logic with classes such as Planet, Ship, Defense, Battle, etc.
- Manage resources (metal and deuterium) to build units and improve technologies.
- Simulate battles with damage calculation, attack probabilities and waste generation.
- Develop a database to store the state of the planet and battles.
- Create a web page to display battle information in HTML format.
- Document the entire process with UML diagrams, tests and user manual.

## Used Technologies

| Category | Tools/Frameworks | Description |
|---|---|---|
| Language | Java | Used to do all the code of the game |
| Data Base | Mysql Server | Used to save the game stats and progres |
| Frontend | HTML,CSS,JS | Web page to put information about the game and the data base reports |
| Version control | GitHub | Used to share the project with the members and have organization in the project |
| Diagrams | Draw.io | Used to do the diagram of classes |
| Development environment | Eclipse and Visual Studio | Eclipse is used to do the code of the game in Java. Visual Studio is used to do the code of the web page. |
| | | |

# Analysis and Design

## Functional and Non-Functional Specifications

### Functional

- Resource Management:
  - Automatic generation of metal and deuterium at regular intervals.
  - Costs associated with unit construction and technology upgrades.

- Unit Construction:
  - Creation of ships (LightHunter, HeavyHunter, BattleShip, ArmoredShip).
  - Construction of defenses (MissileLauncher, IonCannon, PlasmaCannon).

- Technologies:
  - Upgrading attack and defense levels, increasing unit stats.

- Battles:
  - Turn-based combat system with damage calculation, attack probabilities, and debris generation.
  - Detailed battle reports (lost units, resources gained, etc.).

- Data Persistence:
  - Save planet state and battles in an Oracle database.

- Web Interface:
  - HTML page to display battle information (transformed from XML).

### Non-Functional

- Performance:
  - The game must support multiple battles without performance degradation.
- Usability:
  - Clear interface for planet management and battle visualization.

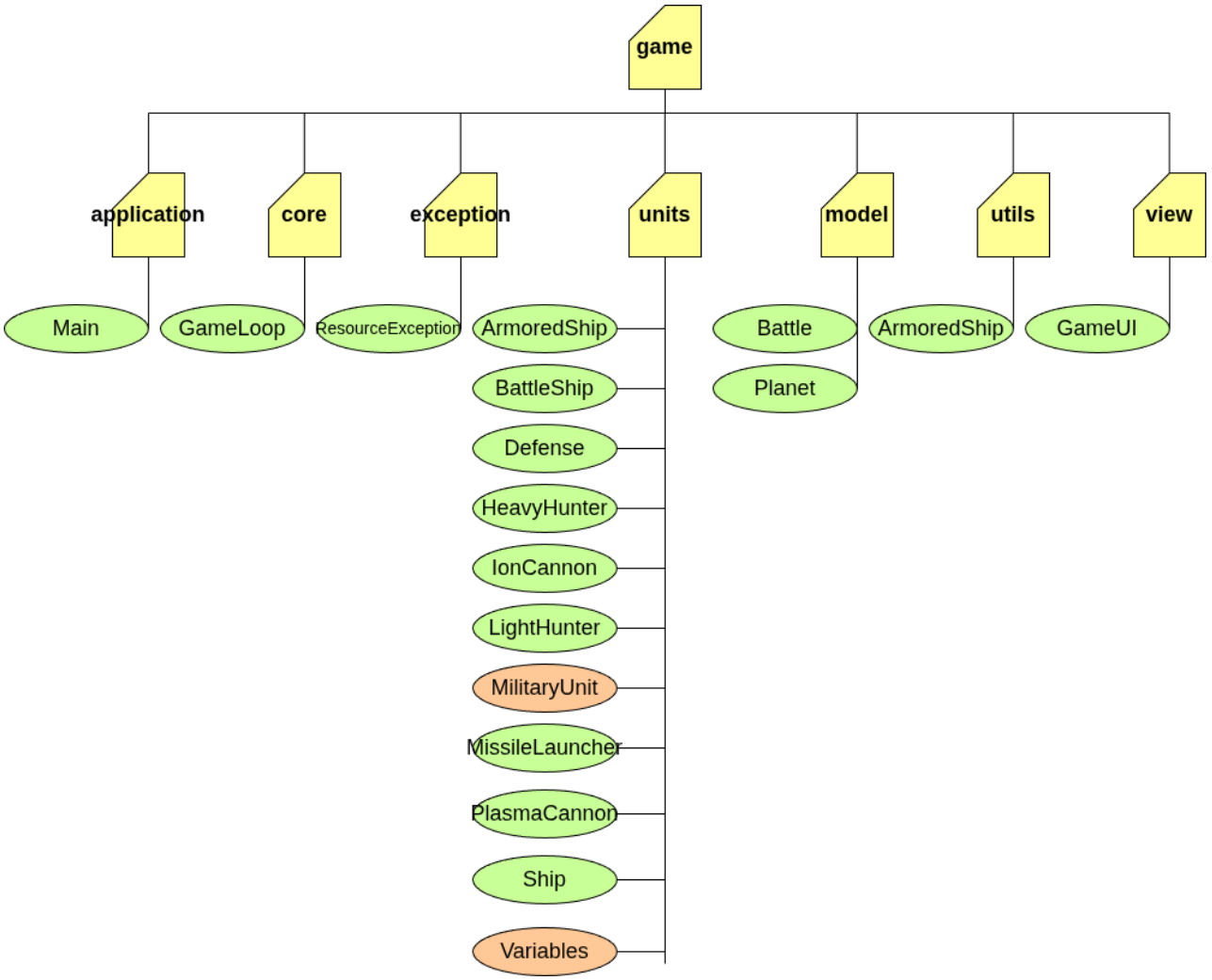
- Security:
  - Secure database access with encrypted credentials.

- Compatibility:
  - Works on different operating systems (Windows, Linux, macOS)

- Documentation:
  - Well-commented code and UML diagrams for easy understanding.

## Game Structure

The structure of the game is divided into the different classes and sections inside the folder “game”. If you see the next image you can see the different sections/folders and inside the sections we have classes in green and interfaces in orange. And all this is the structure of the game, we do it like this because it is better for the organization because a document with 1000 lines of code is not the same as many separate documents with different contents.



## Character Design and Game Mechanics

Military Units

| Type | Role | Characteristics |
|---|---|---|
| LightHunter | Fast and fragile attacker | High chance to attack twice. |
| HeavyHunter | Balanced attack and defense | Moderate damage, medium armor. |
| BattleShip | Heavy ship | Very durable but slow. |
| ArmoredShip | Tank | High armor, low damage. |
| MissileLauncher | Basic defense | Cheap but effective against small ships. |
| IonCannon | Medium defense | Reduces incoming damage |
| PlasmaCannon | Advanced defense | High damage and high chance to eliminate enemy ships. |

Key Mechanics

Turn-based Battles:

- Each unit has a chance to attack based on its type.
- Damage is calculated and debris is generated when a unit is destroyed.

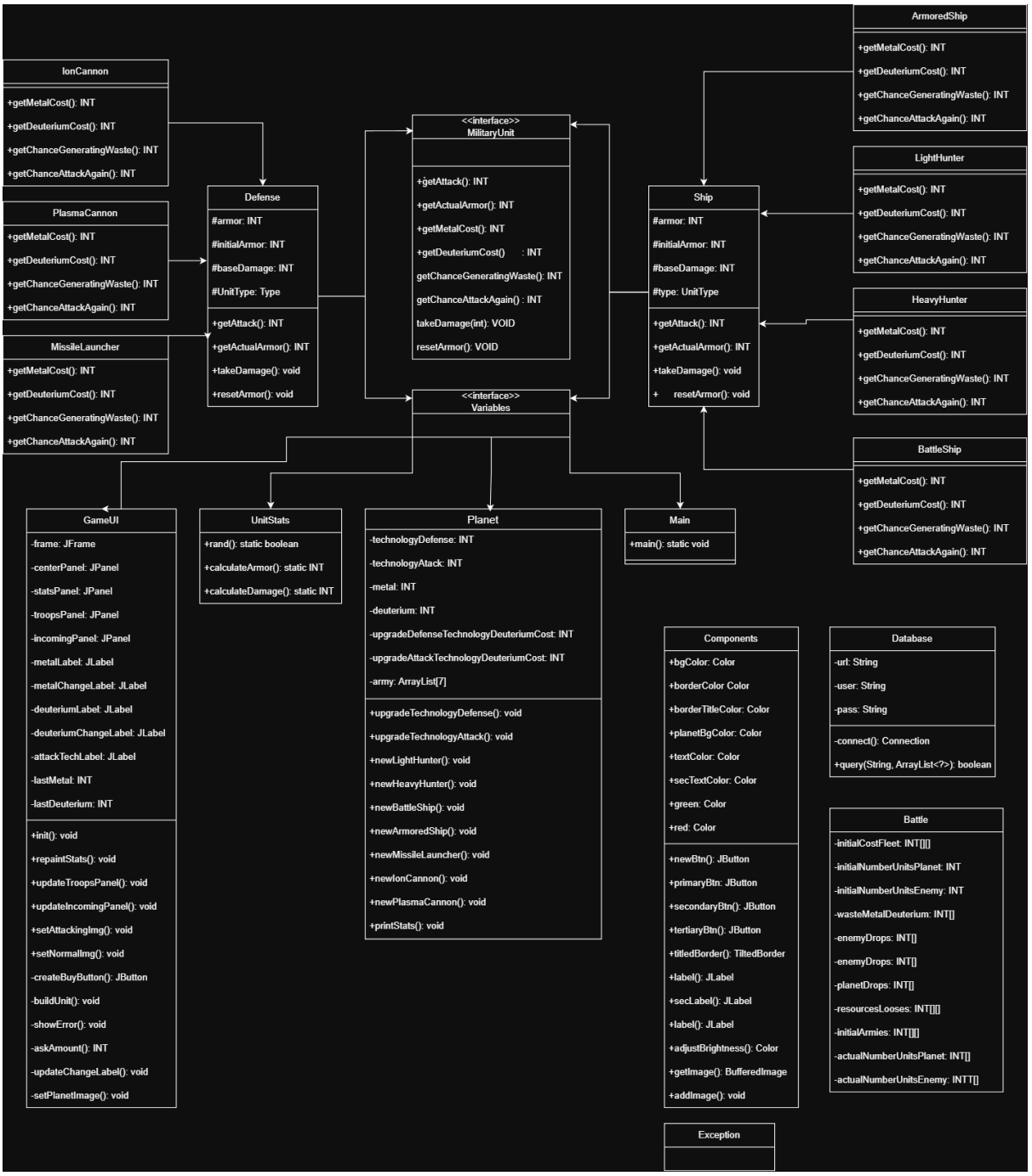
Technologies:

- Each upgraded level increases unit stats.

Resources:

- Earned automatically and spent when building units or upgrading technologies.

# Diagrams


ArmoredShip
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
IonCannon
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
<<interface>>
MilitaryUnit
+ġetAttack(): INT
+getActualArmor(): INT
+getMetalCost(): INT
+getDeuteriumCost() : INT
getChanceGeneratingWaste(): INT
getChanceAttackAgain() : INT
takeDamage(int): VOID
resetArmor(): VOID
LightHunter
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
Defense
#armor: INT
#initialArmor: INT
#baseDamage: INT
#UnitType: Type
+getAttack(): INT
+getActualArmor(): INT
+takeDamage(): void
+resetArmor(): void
Ship
#armor: INT
#initialArmor: INT
#baseDamage: INT
#type: UnitType
+getAttack(): INT
+getActualArmor(): INT
+takeDamage(): void
+ resetArmor(): void
PlasmaCannon
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
HeavyHunter
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
MissileLauncher
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
<<interface>>
Variables
BattleShip
+getMetalCost(): INT
+getDeuteriumCost(): INT
+getChanceGeneratingWaste(): INT
+getChanceAttackAgain(): INT
GameUI
-frame: JFrame
-centerPanel: JPanel
-statsPanel: JPanel
-troopsPanel: JPanel
-incomingPanel: JPanel
-metalLabel: JLabel
-metalChangeLabel: JLabel
-deuteriumLabel: JLabel
-deuteriumChangeLabel: JLabel
-attackTechLabel: JLabel
-lastMetal: INT
-lastDeuterium: INT
+init(): void
+repaintStats(): void
+updateTroopsPanel(): void
+updateIncomingPanel(): void
+setAttackingImg(): void
+setNormalImg(): void
-createBuyButton(): JButton
-buildUnit(): void
-showError(): void
-askAmount(): INT
-updateChangeLabel(): void
-setPlanetImage(): void
UnitStats
+rand(): static boolean
+calculateArmor(): static INT
+calculateDamage(): static INT
Planet
-technologyDefense: INT
-technologyAtack: INT
-metal: INT
-deuterium: INT
-upgradeDefenseTechnologyDeuteriumCost: INT
-upgradeAttackTechnologyDeuteriumCost: INT
-army: ArrayList[7]
+upgradeTechnologyDefense(): void
+upgradeTechnologyAttack(): void
+newLightHunter(): void
+newHeavyHunter(): void
+newBattleShip(): void
+newArmoredShip(): void
+newMissileLauncher(): void
+newIonCannon(): void
+newPlasmaCannon(): void
+printStats(): void
Main
+main(): static void
Components
+bgColor: Color
+borderColor Color
+borderTitleColor: Color
+planetBgColor: Color
+textColor: Color
+secTextColor: Color
+green: Color
+red: Color
+newBtn(): JButton
+primaryBtn: JButton
+secondaryBtn(): JButton
+tertiaryBtn(): JButton
+titledBorder(): TiltedBorder
+label(): JLabel
+secLabel(): JLabel
+label(): JLabel
+adjustBrightness(): Color
+getImage(): BufferedImage
+addImage(): void
Database
-url: String
-user: String
-pass: String
-connect(): Connection
+query(String, ArrayList<?>): boolean
Battle
-initialCostFleet: INT[][]
-initialNumberUnitsPlanet: INT
-initialNumberUnitsEnemy: INT
-wasteMetalDeuterium: INT[]
-enemyDrops: INT[]
-enemyDrops: INT[]
-planetDrops: INT[]
-resourcesLooses: INT[][]
-initialArmies: INT[][]
-actualNumberUnitsPlanet: INT[]
-actualNumberUnitsEnemy: INTT[]
Exception

# Architecture and Development

## Source Structure Description

The *Space Wars* project is developed in Java with a modular structure, organized into several packages:

- application/: Contains the game's entry point (Main.java) and database handling (Database.java).
- exception/: Manages custom errors using ResourceException.java.
- model/: Defines core game elements like planets and battles.
- units/: Includes all military units (ships, defenses), with a base class (MilitaryUnit.java) and shared constants (Variables.java).
- utils/: Provides helper functions, such as unit statistics.
- view/: Manages the graphical user interface and visual assets (GameUI.java, images, etc.).

## Main Components

Planet Class:

- **Attributes**: Tech levels (defense, attack), resources (metal, deuterium), upgrade costs, and an army (array of 7 unit types).
- **Methods**:
  - **upgradeTechnologyDefense() / upgradeTechnologyAttack()**: Upgrade tech levels if there are enough resources.
  - **newUnitX(int n)**: Creates n units of a certain type, if affordable.
  - **printStats():** Displays current planet status including tech, resources, and army composition.

ResourceException:

Custom exception to handle resource shortage errors.

## Ship Class (abstract)

Base class for offensive warships.

- **Attributes**: Armor, base damage.
- **Methods**:
  - **attack()**: Calculates damage based on tech level.
  - **takeDamage(int dmg):** Reduces armor.
  - **getActualArmor()**: Returns armor after damage modifiers.
  - **getMetalCost() / getDeuteriumCost()**: Return build costs.

## MilitaryUnit Interface

Defines methods that all military units (ships and defenses) must implement:

- **attack()**
- **takeDamage(int dmg)**
- **getActualArmor()**
- **canAttackAgain()**
- **generateResources()**

## Ship Types

Concrete subclasses of Ship:

- **LightHunter**
- **HeavyHunter**
- **BattleShip**

- **ArmoredShip**

Each class adjusts damage, armor, and cost based on base values and planet's tech level.

### Defense Class:

Abstract class for defenses, similar to `Ship`, with attributes like armor, base damage, and specific methods.

### MissileLauncher, IonCannon, PlasmaCannon Classes:

Specific defense types that calculate armor and damage based on tech.

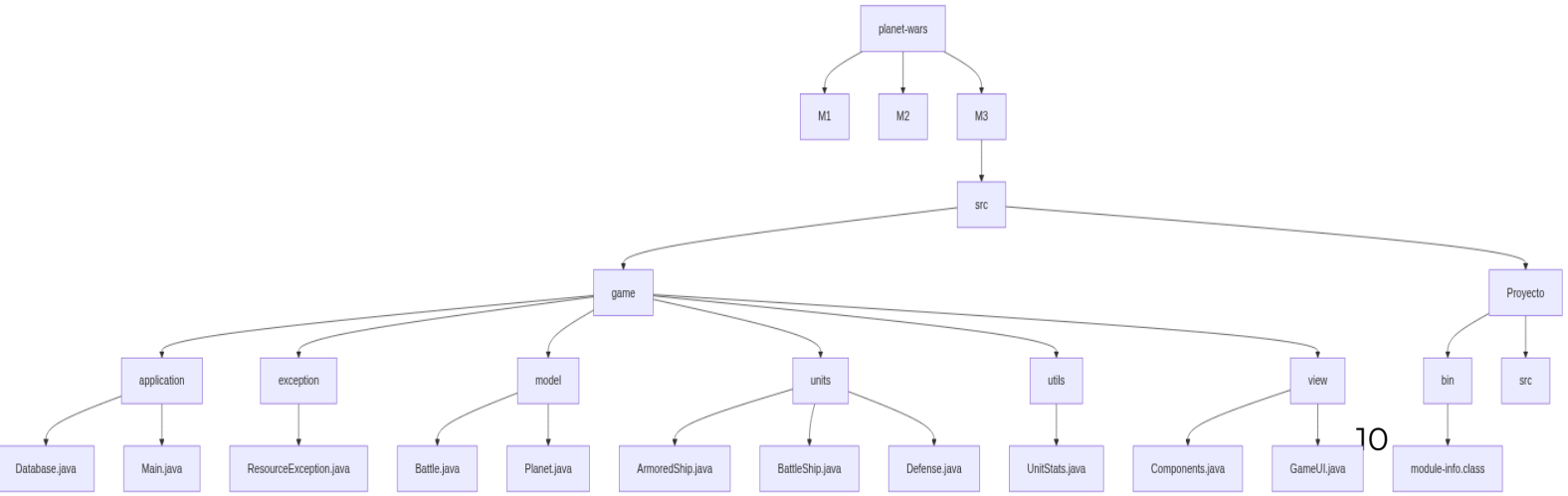
### Variables Interface:

Defines global constants like unit costs, base damage, armor, and probabilities for game control.

### Battle Class:

Manages battles between armies. It calculates attack probabilities, damage, resource losses, and determines the winner based on army effectiveness.

## Project Tree



## Data Flux and Game Events

1. Initialization of the Planet and Its Resources
A Planet object is created with:

- Initial technology levels (defense and attack).
- Initial resources (metal and deuterium).
- Empty initial armies for each unit type (light hunter, heavy hunter, etc.).

2. Unit Generation and Construction
The player decides to create units (fleet or defenses) using methods:

- newLightHunter(n), newHeavyHunter(n), ..., newPlasmaCannon(n).

The planet checks if it has enough resources:

- If not, it throws a ResourceException.
- If it can build some units but not all requested, it creates the maximum possible and throws an exception to notify.

Resources used for construction are deducted.
New units are added to the respective lists within the army[] array.

3. Technology Upgrades
The player can upgrade attack or defense technology:

- Calls upgradeTechnologyDefense() or upgradeTechnologyAttack().

It checks if there are enough resources; if not, it throws a ResourceException.
If upgraded, the cost for the next upgrade increases by a defined percentage.

Upgrades affect:

- Armor of newly created units (increases with defense technology).
- Attack of newly created units (increases with attack technology).

4. Battle Preparation
An instance of the Battle class is created.
The planet's and enemy's fleets are passed to Battle as ArrayList<MilitaryUnit>[].

Initial data stored:

- Resource costs of both fleets (initialCostFleet).
- Initial number of units in each army.
- Variables to control losses and debris generation.

5. Battle Development (in Battle)
The battle progresses in rounds (cycles).

In each round:

- Units from both armies attack based on their probability (attack() and getChanceAttackAgain()).
- Damage each enemy unit receives is calculated (takeDamage()).
- Remaining armor is updated.
- Events are recorded in battleDevelopment for a detailed report.
- At the end of each round, it checks if an army has less than 20% of its initial units:
    - If yes, the battle ends.
- Waste generated (wasteMetalDeuterium) is calculated based on destroyed units.

- Resource losses (resourcesLooses) for both armies are calculated.

6. Outcome and Consequences
If the planet wins:

- It can collect debris generated in the battle to increase resources.

If it loses:

- It loses the resulting debris.

# Testing and Debugging

## Tests Done

We have done different tests which are the following:

- Database: For the database we had to do remote connection tests since we had to use a free host and we had to try connecting from Java and from MySql Workbench to be able to make all the tables and then when we had the data we did tests to see if everything was correct.
- Java: In Java, the tests carried out were to first test all the code without a graphical interface. Once everything worked, we had to incorporate it into the interface and then test it. When everything was correct, we had to connect the data to the database.
- Web: On the website, the tests carried out included testing the entire JavaScript function and then adapting to the phone (responsive web).

## Errors Found

The errors found were the following:

- Database: One of the errors found was the issue of compatibility of some free hosts for the database, which cost us quite a bit to find one without giving errors.
- Java: In Java we have encountered many errors since when programming there are code errors and data issues, also errors when implementing the graphical interface and the data from the database.
- Web: We have found some code errors in the JavaScript issue and errors when adapting to mobile, size issue and little else.

## Development Improvements

We have implemented some improvements such as the graphical interface in Java to make it look simpler or some improvements such as on the web to make it look much better and improve finding a host for the database much more accessible and easier to connect to.

# Graphic Resources

## Fonts

The fonts used have been searched on various websites and some generated with AI, these have been searched and created for the best quality and for the exact sites.

## Integration

The integration of these sources has been, for example, images of the game in the graphical interface, the ships and the background of the planets, for example, or the materials. Then we have the implementation of sources on the web page where we have added images for the logo and the background of the web.

## Fonts Examples

# User Guide

## Instructions To Run The Game

To run the game you must first download the files from github "https://github.com/dps32/planet-wars/tree/main/M3" once you have the files you must run as an application and with Java previously installed the Main.java file with that you should already have the interface open and you would already have the game started

## How to play

Playing is simple, it's about improving your weapons and doing battles and defending yourself against other planets or fleets. You'll gather materials to make various improvements to your planet's defenses and improvements to your fleet, adding more fleets and so on.

# Reflection And Improvements

## Difficulties and how you overcame them.

We have had different difficulties such as the following:

- Java: We have had difficulties with errors and compatibility with the graphical interface and the database as I mentioned previously.
- Database: We had difficulties finding hosts, making connections, and connecting to the database. We easily resolved these difficulties by searching the internet and trying different free hosts.
- Web: Difficulties in previously mentioned topics such as mobile compatibility (responsive) and difficulties in creating JavaScript functions. We have also had difficulties in creating XML documents and implementing them on the website.

## Future Improvements

One of the improvements the project could include would be gameplay enhancements, such as implementing new mechanics or new defenses. New sections could be implemented on the website, such as the "News" section. When we add new features to the game, we can post updates here.