OBJECT ORIENTED PROGRAMMING
FINAL PROJECT

Game Design Document
by Group B

INTRODUCTION

When alien bugs invade the cosmic code repository, only one programmer stands between order and chaos. AslabAttack is a retro arcade shooter inspired by Space Invaders, where players blast waves of descending bugs in a stunning pixel art cosmos—defending the digital universe one line of code at a time!

MEMBERS AND ROLES

Nayla Pramesti Adhina (2406368901) — Gameplay Programmer & Pattern Specialist

- Implementing the Strategy Pattern for interchangeable weapon behaviors (Single Shot vs Spread Shot).
- Developing the Observer Pattern to link the SpaceShip's health state with the UI Heart Bar.
- Managing the Game Loop logic and real-time performance optimization in LibGDX.

Syifa Sarah Nuraini (2406368883) — Fullstack Developer & Backend Architect

- Building the Spring Boot RESTful API for user authentication and leaderboard management.
- Implementing Spring Data JPA to handle CRUD operations for player scores and accounts.
- Connecting the game client to the backend to ensure real-time score synchronization.

Nadira Fayyaza Aisy (2406368933) — Creational Design Architect & Asset Manager

- Implementing the Object Pool Pattern to manage the rapid creation and destruction of bullets and enemies.
- Developing the Factory Method Pattern to instantiate various enemy types like OrcEnemy and GoblinEnemy.
- Handling game assets (textures, sounds) and their integration via OpenGL through LibGDX.

Yohana Indah Nathania Br S. (2406368946) — System Architect & Structural Designer

- Implementing the Singleton Pattern for global management classes like GameManager and SoundManager.
- Applying the Decorator Pattern to handle modular spacecraft upgrades and dynamic attribute additions.
- Maintaining the core OOP Abstraction and Inheritance hierarchy for all game entities to ensure low coupling.

EXECUTIVE SUMMARY

Game Concept

AslabAttack is a retro arcade shooter that combines classic Space Invaders gameplay with modern pixel art aesthetics. Set against a gorgeous cosmic backdrop, players control a lone defender at the bottom of the screen, shooting upward at descending waves of alien bugs. Simple to learn but challenging to master, the game features increasing difficulty, multiple enemy types, and competitive leaderboards.

Target Audience

- Primary: Retro gaming enthusiasts and college students (18-25)
- Secondary: Casual gamers who enjoy pick-up-and-play experiences
- Accessibility: Instant understanding - move left/right, shoot up

Genre

- Arcade Shooter / Space Invaders-like / Score Attack
- Single-player with online leaderboards

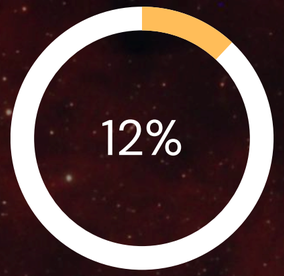
Unique Selling Points

- Stunning Pixel Art: Beautiful cosmic background with vibrant, hand-crafted sprites
- Programming Theme: Bugs as enemies, code-themed power-ups and scoring
- Progressive Difficulty: Wave-based system that gets increasingly challenging
- Retro Feel, Modern Polish: Classic arcade gameplay with smooth controls and effects
- Competitive Leaderboards: Global ranking system with authentication

Game Pillar

IMMEDIATE ACTION

No complex menus or tutorials needed. The moment the game starts, players know exactly what to do: move and shoot.

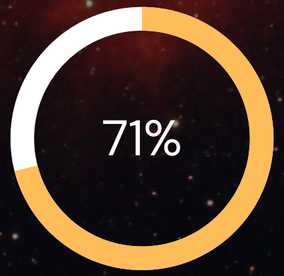


ESCALATING CHALLENGE

Each wave introduces new patterns, faster enemies, and tougher bugs. Mastery comes from learning patterns and perfecting timing.

SATISFYING FEEDBACK

Every shot, every hit, every explosion feels impactful. Visual and audio feedback creates a deeply satisfying gameplay loop.



ESCALATING CHALLENGE

The game pays homage to arcade classics while adding its own unique personality through gorgeous pixel art and programming humor.

GAMEPLAY MECHANICS

1. PLAYER MOVEMENT

- HORIZONTAL MOVEMENT ONLY - LEFT/RIGHT ARROW KEYS OR A/D
- SMOOTH ACCELERATION WITH RESPONSIVE CONTROLS
- SCREEN BOUNDARIES - CAN'T MOVE OFF EDGES
- MOVEMENT SPEED: 200 PIXELS/SECOND BASE SPEED

2. SHOOTING SYSTEM

- AUTO-FIRE OR MANUAL FIRE (SPACEBAR) - PLAYER CHOICE IN SETTINGS
- BULLET TYPE: SINGLE STRAIGHT PROJECTILE UPWARD
- FIRE RATE: 5 SHOTS PER SECOND (200MS COOLDOWN)
- BULLET SPEED: 400 PIXELS/SECOND
- ONE BULLET ON SCREEN AT A TIME (CLASSIC MODE) OR MULTIPLE (MODERN MODE)

3. ENEMY BEHAVIOR

- FORMATION MOVEMENT: ENTIRE GRID MOVES LEFT/RIGHT TOGETHER
- DESCENT PATTERN: EACH COMPLETED HORIZONTAL SWEEP, FORMATION DROPS DOWN
- SPEED INCREASES: AS BUGS ARE ELIMINATED, REMAINING BUGS MOVE FASTER
- SHOOTING: RANDOM BUGS FIRE PROJECTILES DOWNWARD AT INTERVALS
- LANDING CONDITION: IF ANY BUG REACHES BOTTOM = GAME OVER

4. COLLISION SYSTEM

- PLAYER BULLET HITS BUG: BUG DESTROYED, SCORE INCREASES
- ENEMY BULLET HITS PLAYER: LOSE 1 LIFE
- BUG REACHES BOTTOM: INSTANT GAME OVER
- SHIELDS: OPTIONAL BARRIERS PLAYER CAN HIDE BEHIND (DESTRUCTIBLE)

GAMEPLAY MECHANICS

5. SCORING SYSTEM

NULL POINTERS: 10 PTS
LOGIC ERRORS: 20 PTS
MEMORY LEAKS: 40 PTS
STACK OVERFLOWS: 40 PTS
FLYING SYNTAX ERRORS: 100-300 PTS (BASED ON WHEN HIT)
BOSS BUGS: 1000 PTS
WAVE CLEAR BONUS: WAVE# × 100 PTS
PERFECT WAVE (NO HITS TAKEN): +500 PTS
COMBO MULTIPLIER: 1.5X AFTER 10 CONSECUTIVE HITS

6. LIVES SYSTEM

- START WITH 3 LIVES
- EXTRA LIFE AWARDED EVERY 10,000 POINTS
- MAX 5 LIVES AT ONCE
- HIT BY ENEMY PROJECTILE = -1 LIFE
- 0 LIVES = GAME OVER

7. POWER-UPS (RANDOM DROPS)

- RAPID FIRE: INCREASED FIRE RATE FOR 10 SECONDS
- SPREAD SHOT: FIRE 3 BULLETS AT ONCE FOR 10 SECONDS
- SHIELD: TEMPORARY INVINCIBILITY FOR 5 SECONDS
- SLOW TIME: ENEMIES MOVE AT 50% SPEED FOR 8 SECONDS
- SCORE MULTIPLIER: 2X POINTS FOR 15 SECONDS

8. WAVE SYSTEM

- 15 PRE-DESIGNED WAVES WITH INCREASING DIFFICULTY
- WAVE 16+: PROCEDURAL GENERATION, ENDLESS SURVIVAL
- BETWEEN WAVES: 3-SECOND BREAK TO PREPARE
- WAVE COMPLETE: ALL BUGS ELIMINATED
- BOSS WAVES: EVERY 5 WAVES (5, 10, 15, 20...)

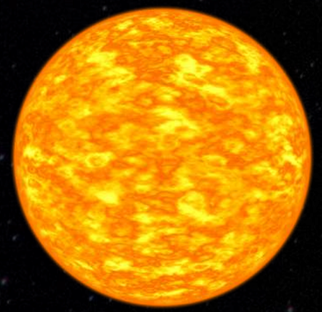
HOW TO PLAY:

- Objective: Survive as long as possible and eliminate alien swarms to achieve the highest score.
- Movement & Combat: Use the controls to navigate the ship, dodge incoming enemies, and fire projectiles to destroy them.
- Health System: The player starts with 3 hearts (life bar). The game ends when all lives are lost due to collisions or enemy fire.
- Upgrades: Collect power-ups to increase the "rocket lethal capacity" and unlock new weapon behaviors.
- Weapon Switching: Thanks to the Strategy Pattern, players can dynamically switch between different attack modes, such as single laser or spread shots

USER INTERFACE (UI)

MINIMAL DESIGN PHILOSOPHY:

- The game canvas displays a pixelated, retro arcade aesthetic with a rich purple nebula backdrop, complemented by cyan-blue planetary rings in the corners and scattered star particles. Red square enemies create stark contrast against the cosmic scenery.
- The design uses a cohesive space theme with deep purples, bright cyans, vibrant reds for enemies, and pink/white for UI elements and projectiles, creating a nostalgic yet modern retro-futuristic feel.



MAIN MENU:

- A prominent grey banner with a bold black outline contains the pixelated purple title, establishing a retro-arcade aesthetic. Below, a vibrant pink "PLAY" button serves as a high-contrast call-to-action, ensuring a simple and intuitive user experience.

USER INTERFACE (UI)

TOP BAR:

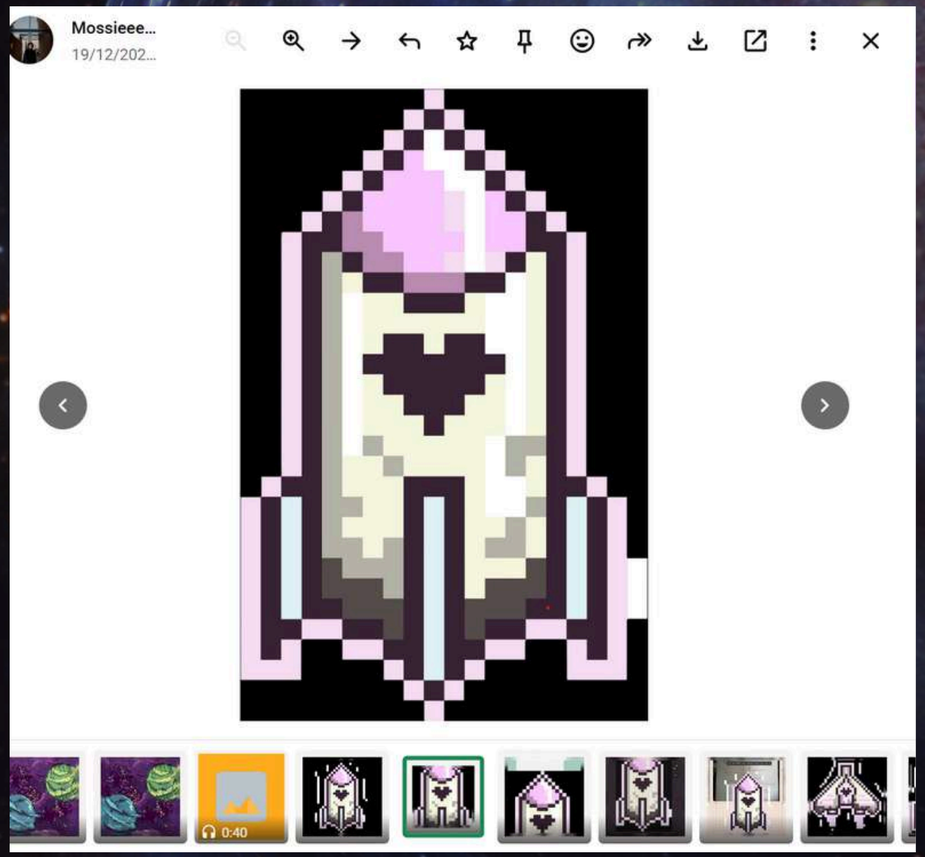
- Score: Current player score (left side)
- Lives: Heart or ship icons showing remaining lives
- Wave Counter: Current wave number

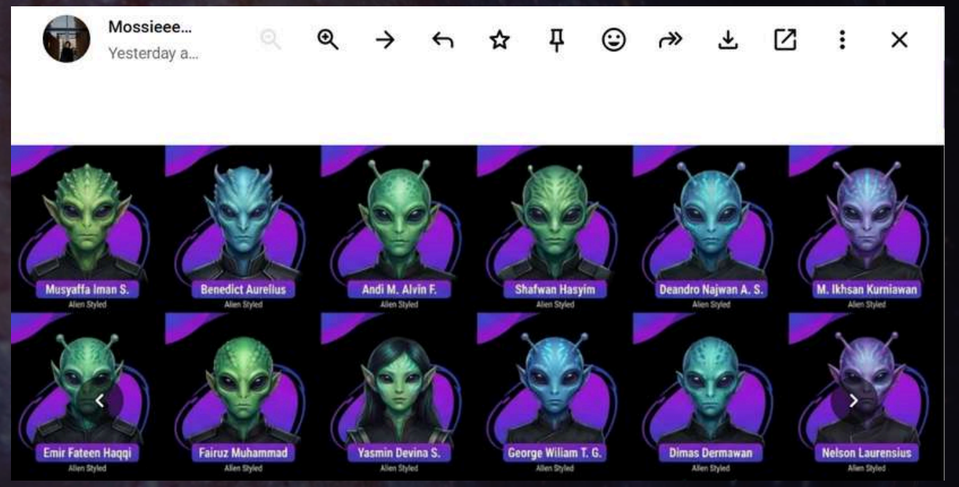
GAME OVER/VICTORY/LEVEL UP:

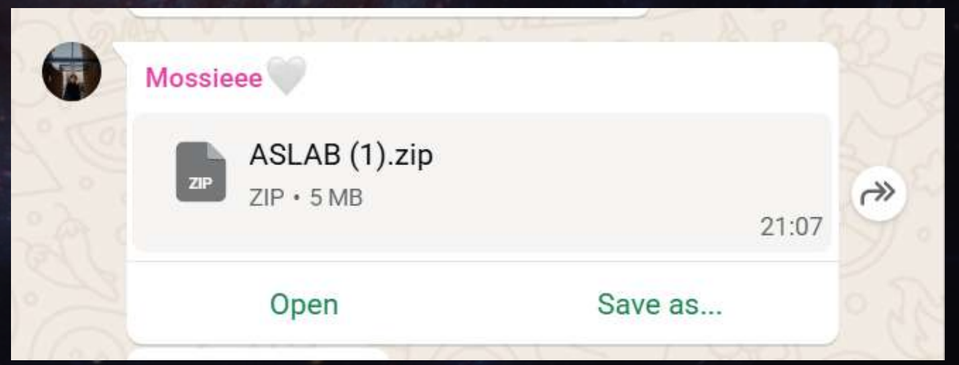
- The game canvas displays a pixelated, retro arcade aesthetic with a rich purple nebula backdrop, complemented by cyan-blue planetary rings in the corners and scattered star particles. Red square enemies create stark contrast against the cosmic scenery.
- The design uses a cohesive space theme with deep purples, bright cyans, vibrant reds for enemies, and pink/white for UI elements and projectiles, creating a nostalgic yet modern retro-futuristic feel.

ASSET CREDITS

We made it by ourself, credit to moss :D







IMPLEMENTATION MODULES

01. Modules 1-3: OOP Fundamentals (Class, Object, UML, Encapsulation, Inheritance, Polymorphism, & Abstraction)

- Implementation: The entire project is built on these principles.
- Class & Object: Seen in the definitions of PlayerScore, User, and various game entities.
- Inheritance: Used extensively in the enemy system. For example, the GoblinsEnemy and BossEnemy classes extend Enemy.
- Abstraction: Shown in the abstract class Enemy, which defines abstract methods like update and render that every specific enemy type must implement.

02. Module 4: Generic Type, Collection, and Iterator Pattern

- Implementation: Visible through the use of java.util.List and ArrayList to manage multiple objects.
- In GameScreen or the Managers, lists are used to store bullets and enemies.
- Generics like List<Enemy> ensure type safety, making sure only enemy objects are stored in that specific collection.

IMPLEMENTATION MODULES

03. Modules 5 & 6: Spring Boot, RESTful API, Spring Data JPA & CRUD

- Implementation: Found within the backend folder.
- Controller: AuthController and ScoreController handle HTTP requests (POST/GET) for registration, login, and score submission.
- JPA & CRUD: The UserRepository and PlayerScoreRepository interfaces (which extend JpaRepository) are used for database operations, such as saving new users or fetching high scores.

04. Module 7: LibGDX, Game Loop, and Singleton Pattern

- Implementation:
- LibGDX & Game Loop: The code uses the LibGDX framework. The render(float delta) method in various classes (like GameScreen) is part of the Game Loop that runs continuously to update logic and graphics.
- Singleton Pattern: Implemented in the GameManager. This class has a private static GameManager instance and a getInstance() method, ensuring only one manager controls scores and levels throughout the game sessio

IMPLEMANTATION MODULES

Module 8: Factory Method & Object Pool Pattern

Implementation:

- Factory Method: Seen in EnemyFactory, GoblinEnemyFactory, and OrcEnemyFactory. This pattern is used to create enemy objects without specifying their exact class in the main game logic.
- Object Pool: Found in BulletPool. Instead of creating a new bullet object (which is memory-intensive) every time a shot is fired, the code reuses existing objects from the pool and reactivates them.

Module 9: Observer & Command Pattern

Implementation:

- Observer Pattern: Visible through the Observable and Observer interfaces. The SpaceShip class acts as the subject that notifies observers (like the UI Health Bar) whenever its status changes (e.g., when taking damage).

IMPLEMENTATION MODULES

Module 10: Strategy & State Pattern

- Implementation:
- Strategy Pattern: Used in the shooting system (ShootingStrategy). There are SingleShotStrategy and SpreadShotStrategy. The ship can easily switch firing patterns just by swapping the strategy object.
- State Pattern: Seen in how the GameScreen manages game states (e.g., PLAYING, PAUSED, or GAME_OVER), which determines what logic should be executed at any given time.

Module 11: Template Method & Facade Pattern

- Implementation:
- Template Method: Found in the Enemy base structure. The overall flow of how an enemy moves and takes damage is defined in the base class, while specific movement details are left to the sub-classes.
- Facade Pattern: Seen in how the SpaceShooterBackendApplication or main Manager classes simplify complex interactions between systems (audio, input, and rendering) into a single, easy-to-use interface.

LINK GITHUB

https://github.com/Lilyof1hevalley/FINPRO_OOP_GROUP-B.git

Thank You

ORANG KEREN

IZIIIINNNNNN

VHOOOOMMMMMM