Tank War Game

Project Overview

In this project, you will design and implement a 2D tank war game using **JavaFX (required)**. The game is inspired by the classic "Battle City" game (See the video here), where players control a tank, avoid enemy attacks, and destroy enemy tanks. You will use Object-Oriented Programming (OOP) concepts and apply **Design Patterns** to build a modular and scalable game.

The game should feature the following:

- **Player-Controlled Tank:** Move, fire missiles, and avoid enemy tanks and walls.
- **Enemy Tanks:** Move and attack the player's tank. You can implement a simple AI to control their behaviors. When game starts, by default, there are 6 enemy tanks.
- **MedPacks:** Heal tanks (Both player and enemy tanks) when collected.
- **Missiles:** Fired by both the player's tank and enemy tanks, they deal damage to other tanks. For simplicity, missiles will not damage the walls and will be "absorbed" by walls.
- **Walls:** Serve as obstacles and are indestructible.
- **Explosions:** Trigger when a tank is destroyed.

I provided some images you can use for explosions, tanks, and missiles.

The game is designed to focus on core OOP principles (encapsulation, inheritance, polymorphism) and common **design patterns** (such as Factory, Strategy, Singleton, Observer, etc.).

Project Objectives

By the end of this project, students will:

1. Apply **Object-Oriented Design (OOD)** principles to model real-world objects (e.g., Tank, Wall, Missile, etc.).
2. Implement a **Graphical User Interface (GUI)** in JavaFX, the latest Java GUI technology.
3. Use **Design Patterns** to promote flexibility, reusability, and maintainability.
4. Understand the separation of concerns by following good software design practices.
5. Develop basic game logic, AI, and mechanics.

Game Features and Requirements

1. Core Classes/Abstractions

Students should create the following classes based on OOP principles:

- **Tank:** Represents both the player-controlled and enemy tanks. It can move in four directions (up, down, left, right), fire missiles, and take damage. Player and enemy tanks should behave similarly but may have differences in their behavior (e.g., player-controlled vs. AI-controlled).
- **Missile:** Represents a projectile fired by tanks. It moves in a straight line and can damage tanks.
- **Wall:** Cannot be destroyed or crossed by tanks and missiles.
- **MedPack:** When collected by a tank, it restores full health.
- **Explosion:** Triggered when tanks are destroyed.
- **Direction:** Represents the tank's movement direction (up, down, left, right).

2. Game Mechanics

- **Tank Movement**: The player's tank should be controlled via keyboard input (arrow keys). Tanks should be able to move freely, except when blocked by a wall or the edge of the screen.
- **Missiles**: The player and enemy tanks can fire missiles. The missiles move in the direction the tank was facing when fired.

- **Enemy AI**: Enemy tanks should have basic AI to move around and fire at the player's tank.
- **Health System**: Tanks should have health points. When health reaches zero, the tank is destroyed and the animation of explosion should be played.
- **Winning/Losing Conditions**: The player wins if they destroy all enemy tanks and loses if their tank is destroyed.

3. Design Patterns

- **Factory Pattern**: For creating different types of objects (Tanks, Walls, Missiles, etc.). This ensures scalability, where new types of tanks or walls can be easily added.
- **Strategy Pattern**: For different movement behaviors (e.g., player-controlled vs. AI-controlled tanks). This allows easy modification of tank behavior.
- **Observer Pattern**: To handle game events, such as tanks being destroyed or health pickups. For example, the game can notify the GUI to update when a tank is destroyed.
- **Singleton Pattern**: For managing shared resources, such as the game map or a global state like score or remaining lives.
- **More**...

4. Game GUI

The game should have a simple **JavaFX** GUI for displaying the game. The interface should include:

- **Game Area:** Where the action happens (tank movement, shooting, etc.).
- **Health Bar:** Displays the player's tank health.
- **Score/Lives Counter**: Shows how many enemies are left and how many lives the player has.

5. Bonus Features

To push advanced students, you can offer additional optional features:

- **Destructible walls**.
- **Advanced AI**: Implement smarter AI for enemy tanks, like seeking the player or coordinating attacks.

Deliverables

1. **Design Document (UML Diagrams) in Mermaid.js notation**:
    - A class diagram showing how different game objects interact (Tank, Wall, Missile, etc.).
2. **Java Source Code**:
    - Well-organized and modular codebase using OOP and design patterns.
    - JavaFX GUI for game display and user interactions.
3. Manual
    - A brief description of how to start the game.
4. A Demo Video

Evaluation Criteria

- **Code Quality (40%)**: Proper use of OOP principles, modularity, readability, and use of design patterns.
- **Functionality (40%)**: Correct implementation of game features (movement, shooting, AI, etc.).
- **GUI (20%)**: User-friendly interface with proper display of game objects and game state.