NanoWar: Technical and Architectural Report

IIT Tirupati - Pranay, Neha, Srivally, Mounika

January 11, 2025

1 Executive Summary

NanoWar is an educational 2D action game developed in Unity, aimed at promoting awareness of viral infections and their impact on human health. It aligns with UN Sustainable Development Goals (SDGs) for Good Health and Well-being (SDG 3) and Quality Education (SDG 4). Players control a nanobot navigating the human bloodstream, battling viruses while learning about microbiology. The game features progressive levels, interactive storyboards, and educational content, built using modular C# scripts, Unity's Spline system for enemy paths, and asset management for cross-platform compatibility.

This report provides a comprehensive technical and architectural overview, including system design, code structure, implementation details, design pattern analysis, team contributions, and educational impact.

2 Project Overview

2.1 Game Concept and Objectives

- **Theme**: Futuristic nano-scale world inside the human body; players operate a nanobot to eliminate viruses, avoid white blood cells (WBCs), and collect health boosts.
- **Educational Integration**: Gameplay interwoven with facts on virus types, behaviors, and health impacts.
- **Target Audience**: All ages — simple controls for casual play, increasing difficulty for engagement.
- **SDG Alignment**: Simulates viral damage to promote health awareness and educates on microbiology via dedicated info sections.

2.2 Key Features

1. **User Interface**: Intuitive menus, health/fuel bars, score/time counters using Unity UI and TextMeshPro.
2. **Interactive Storyboard**: Narrative scenes (`StoryBoard1-6.unity`) with images and audio.
3. **Virus Education**: Dedicated scenes (`Virus1-4.unity`) with engaging visuals and facts.

4. **Gameplay Mechanics**:
   - 4 progressive levels (`Level1-4.unity`).
   - Nanobot movement, projectile shooting, collision detection.
   - Power-ups (health boosts), timers, scoring.
5. **In-Game Elements**: Health/fuel bars, spline-based enemy AI, parallax scrolling.
6. **Audio/Visuals**: Persistent background music, sound effects, sprite-based art.

2.3 Development Environment

- **Engine**: Unity (2022+ compatible — uses Splines and new Input System).
- **Language**: C#.
- **Asset Organization**: Folders for Animations, Art, EnemyTypes (ScriptableObjects), Prefabs, Scenes, Scripts, Settings, WeaponStrategies.
- **Cross-Platform**: PC and mobile support.
- **Version Control**: Git.
- **Packages**: Unity Input System, Splines, Eflatun.SceneReference, custom Utilities.

3 Team Contributions

The project was a collaborative effort by four team members, each specializing in key areas:

- **Pranay**: Development of game levels and scripting functionalities; integration of gameplay mechanics and level progression balance.
- **Neha**: Creation of the interactive visual storyboard; designing narrative elements and ensuring SDG alignment.
- **Mounika**: Design and development of UI and graphics; implementation of visual elements for player engagement.
- **Srivally**: Research and inclusion of detailed virus information; ensuring scientific accuracy and educational value.

This division allowed focused expertise while maintaining cohesive integration.

4 Architectural Design

NanoWar follows a modular, component-based architecture typical of Unity games, emphasizing separation of concerns, reusability, and scalability.

4.1 High-Level Architecture

- **Scene Management**: Multiple scenes (MainMenu, Loading, Levels 1–4, Storyboards, Virus Info, GameOver) with smooth transitions via `Loader.cs`.
- **Entity Behaviour**: Components attached to GameObjects (e.g., `PlayerController`, `Enemy`, `Projectile`).
- **Input**: Unity Input System processed through `InputReader.cs`.
- **Game Loop**: Centralized in `GameManager.cs` (Singleton).
- **Rendering**: 2D orthographic camera with parallax (`ParallaxController.cs`).
- **Audio**: Persistent via `Music.cs` (Singleton + DontDestroyOnLoad).
- **Data**: ScriptableObjects for configurable enemy types and weapon strategies.

4.2 Design Patterns Employed

The codebase deliberately employs several classic design patterns to achieve flexibility, maintainability, and extensibility. Each pattern is directly evidenced in the provided source code.

1. **Builder Pattern** (`EnemyBuilder.cs`)
   A fluent builder class constructs complex enemy GameObjects step-by-step:

   ```
   public class EnemyBuilder {
       public EnemyBuilder SetBasePrefab(GameObject prefab) { ... }
       public EnemyBuilder SetSpline(SplineContainer spline) { ... }
       public EnemyBuilder SetSpeed(float speed) { ... }
       public GameObject Build() { /* instantiates, adds SplineAnimate, configures
   }
   ```

   Benefits: Decouples enemy configuration from instantiation; allows easy variation without multiple constructors.

2. **Factory Pattern** (`EnemyFactory.cs`)
   Centralizes enemy creation and integrates the Builder:

   ```
   public GameObject CreateEnemy(EnemyType enemyType, SplineContainer spline) {
       EnemyBuilder builder = new EnemyBuilder()
           .SetBasePrefab(enemyType.enemyPrefab)
           .SetSpline(spline)
           .SetSpeed(enemyType.speed);
       GameObject enemy = builder.Build();
       // Additional setup (destruction callback)
       return enemy;
   }
   ```

   Benefits: Hides construction complexity; enables future subclassing.

3. **Strategy Pattern** (`WeaponStrategy.cs`, `SingleShot.cs`, `Weapon.cs`)
   Defines interchangeable firing algorithms:

```
public abstract class WeaponStrategy : ScriptableObject {
    public abstract void Fire(Transform firePoint, LayerMask layer);
}
[CreateAssetMenu(...)]
public class SingleShot : WeaponStrategy {
    public override void Fire(...) { /* instantiate single projectile */ }
}
public abstract class Weapon : MonoBehaviour {
    [SerializeField] protected WeaponStrategy weaponStrategy;
    public void SetWeaponStrategy(WeaponStrategy strategy) { ... }
}
```

Benefits: New weapons can be added as ScriptableObjects without modifying core classes.

4. **Singleton Pattern** (`GameManager.cs`, `Music.cs`)
   Ensures one instance for global state.
5. **Observer Pattern** (via `UnityEvent`)
   For loose coupling in enemy destruction notifications.
6. **Model-View-Controller (MVC)-like Structure**

   - **Model**: ScriptableObjects (`EnemyType`, `WeaponStrategy`).
   - **View**: UI, sprites, particle effects.
   - **Controller**: `PlayerController`, `EnemySpawner`, `GameManager`.

4.3 System Components and Data Flow

- **Player System**: `Player.cs` (health/fuel), `PlayerController.cs` (movement), `PlayerWeapon.cs` (firing).
- **Enemy System**: Spline-based movement, random spawning, auto-firing.
- **Data Flow**: Input → `InputReader` → Controllers → Spawning (Factory/Builder) → Update loops → Collisions → `GameManager`.

5 Educational Impact and SDG Alignment

NanoWar uniquely blends entertainment with education:

- Illustrates viral damage to human cells in real-time gameplay.
- Provides accurate microbiology facts in dedicated scenes (researched and verified).
- Promotes preventive health awareness subtly through narrative and mechanics.
- Suitable for classrooms, health campaigns, or self-learning.
- Aligns with SDG 3 (health) by simulating immune response and SDG 4 (education) via engaging content delivery.

6 Technical Implementation Details

6.1 Key Scripts Breakdown

Script	Purpose	Key Features
`CameraController.cs`		
Vertical camera scrolling	Constant speed movement after initial player follow	Fluent enemy construction
`EnemyBuilder.cs`		
Configures prefab, spline, speed	Enemy instantiation	Integrates Builder and adds callbacks
`EnemyFactory.cs`		
Periodic spawning	Random type/spline, max limit, cleanup	Global state
`GameManager.cs`		
Singleton, score, game over logic	Abstract firing behaviour	ScriptableObject base for strategies
`WeaponStrategy.cs`		
Concrete strategy	Single projectile firing	UI updates
`SingleShot.cs`		
`PlayerHUD.cs`		
Health/fuel bars, score, formatted time	Bullet logic	Movement, VFX, damage on hit
`Projectile.cs`		

6.2 Asset Organization

- **Art**: Sprites (viruses, nanobot, UI, storyboards).
- **Prefabs**: Reusable objects (NanoBot, Viruses, Projectiles).
- **Scenes**: 15+ scenes for menus, levels, education.
- **EnemyTypes / WeaponStrategies**: Data-driven ScriptableObjects.

7 Challenges and Solutions

- Smooth curved enemy movement → Unity Splines + `SplineAnimate`.
- Reliable collisions → Dual `OnCollisionEnter`/`OnTriggerEnter`.
- Persistent music → Singleton + `DontDestroyOnLoad`.
- Player bounds → Dynamic clamping in `PlayerController`.
- Extensibility → Design patterns (Builder, Factory, Strategy).

8 Testing and Deployment

- Manual testing in Unity Play Mode with debug logs.

- Stable performance due to lightweight 2D design.
- Deployment via Unity Build Settings (PC/Windows primary; mobile ready).

9 Future Enhancements

- Additional levels and virus types.
- New weapon strategies (multi-shot, homing).
- Leaderboards and persistent saves.
- Touch-optimized controls.
- Interactive educational quizzes.