DefenCLT: Game Design Document

1. Introduction

1.1 Purpose
This document outlines the design for **DefenCLT**, a tower-defense style game built in Unity. It is intended to guide development, ensure alignment among team members, and provide clear specifications for both gameplay and technical implementation.

1.2 Scope
DefenCLT tasks players with protecting the University of North Carolina at Charlotte (UNCC) campus from invading food delivery robots by strategically placing turrets on a grid-based map. Core systems include:

- **Unity** game engine for gameplay and rendering
- **Firebase Authentication** for user login/registration
- **Firestore** database for storing and displaying leaderboards

2. Game Concept

2.1 Genre & Theme

- Genre: Tower Defense
- Theme: Campus under siege by food delivery robots

2.2 Target Audience
Casual gamers and students who enjoy strategy and tower-defense mechanics.

3. Gameplay Overview

3.1 Objective
Prevent waves of robots from reaching the campus center by placing and upgrading turrets.

3.2 Core Loop

1. Authenticate and start a new session.
2. Load map and initial currency.
3. Place turrets on designated tiles.
4. Trigger next wave: robots spawn and follow predefined paths.
5. Turrets automatically target and fire bullets at robots.
6. Robots that reach the end reduce player lives.
7. Player earns currency for each robot destroyed.
8. Repeat until all waves are completed or lives reach zero.
9. Submit score to Firestore leaderboard.

3.3 Win/Lose Conditions

- **Win:** Survive all waves.
- **Lose:** Lives drop to zero before final wave.

4. Game Mechanics

4.1 Turret Placement

- Grid-based tilespace limits placement to valid squares.
- Each tile has a `Tile` component that tracks occupancy.
- Placement Manager handles player input, currency deduction, and turret instantiation.

4.2 Turret Behavior

- Prefabs include parameters: range, fire rate, damage.
- `Turret` script rotates toward the nearest robot in range and spawns `Bullet` prefabs.

4.3 Enemy AI

- Robots follow waypoints defined in the Map object.
- `RobotManager` spawns robots per wave configuration, tracks active robots, and handles removal on death or arrival.

4.4 Wave & Economy System

- `GameStatsManager` stores current wave index, currency, lives, and spawn counts.
- Wave configurations (e.g., number of robots, spawn interval) stored in a ScriptableObject or JSON.
- Currency awarded per kill; used for building/upgrading turrets.

5. Technical Architecture

5.1 Project Structure

```
Assets/
  Scenes/
    Main.unity
    Level 1
  Prefabs/
    Turret.prefab
    Robot.prefab
    Bullet.prefab
  Scripts/
    Managers/
```

PlacementManager.cs
RobotManager.cs
GameStatsManager.cs
Gameplay/
Turret.cs
RobotAI.cs
Bullet.cs
Services/
AuthService.cs
LeaderboardService.cs

5.2 Scenes & Objects

- **Main Scene:** Contains the Map (tilespace), UI Canvas, Managers.
- **UI Canvas:** Login panel, in-game HUD, post-game leaderboard.
- **Level 1:** The map layout with tilemap and nodes.

6. Firebase Integration

6.1 Authentication Flow

- `AuthService` manages sign-up, login, and logout using Firebase Auth SDK.
- On successful login, user's display name and UID are cached for session.

6.2 Leaderboard

- `LeaderboardService` submits scores to Firestore collection `leaderboard` with fields: `uid`, `displayName`, `score`, `timestamp`.
- Top N scores fetched and displayed in UI on game over.

7. User Interface (UI)

7.1 Login & Main Menu

- Fields: Email, Password, Login/Register buttons.
- On login success: load Main Scene.

7.2 In-Game HUD

- Displays lives, currency, current wave.
- Buttons: Next Wave, Pause.

7.3 Leaderboard Screen

- Table showing top scores with names and wave reached.

8. Art & Audio Assets

8.1 Visuals

- Placeholder sprites for turrets, robots, bullets.
- Map background: aerial view of campus.

8.2 Audio

- SFX: turret firing, robot destruction, button clicks.
- BGM: looped track for in-game ambiance.

9. Map & Level Design

9.1 Map Layout

- Single predefined map with fixed path waypoints.
- Tilespace overlay showing buildable zones.

9.2 Wave Progression

- 10 waves of increasing difficulty.
- Example: Wave 1: 5 robots; Wave 10: 30 robots with 2 sprint bots.

10. Testing & QA

- Unit tests for Managers (e.g., wave spawning logic).
- Playtesting to balance turret stats and wave difficulty.
- Authentication and leaderboard integration tests.

11. Future Enhancements

- Multiple maps and randomized layouts.
- Turret upgrades and special abilities.
- Power-ups and boss waves.
- Mobile input optimization.