

TECHNICAL DESIGN DOCUMENT (TDD) PROJECT: SIEGE OF GONDOR

VERSION: 1.2.0

DATE: January 19, 2026

COURSE: SENG479 - Game Programming

AUTHORED BY:

- Ahmet Can Cengiz
- Ahmet Said Kuruoğlu
- Anday Turgut
- Bartu Turgut
- Salih Aydos
- Mustafa Kayalica

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY.....	3
2. TECHNICAL ARCHITECTURE.....	4
3. GAMEPLAY SPECIFICATIONS.....	6
4. PROGRESSION & GOALS.....	9
5. CONTROLS & USER INTERFACE.....	10
6. ASSET ATTRIBUTION & LICENSING.....	11
7. DESIGN CHALLENGES & SOLUTIONS.....	12

1. EXECUTIVE SUMMARY

1.1 Project Scope

Siege of Gondor is a real-time tactical Tower Defense application developed natively in C/C++. The project aims to demonstrate proficiency in low-level game programming concepts, eschewing commercial game engines (e.g., Unity, Unreal) to showcase manual memory management, custom collision detection algorithms, and state management.

1.2 Objective

The user acts as the strategic commander of Gondor, tasked with preventing enemy entities from traversing a fixed path to the "Main Tower" coordinate. The application defines a victory condition upon the successful clearance of 20 generated waves, and a failure condition if the base integrity reaches zero.

2. TECHNICAL ARCHITECTURE

2.1 Development Environment

- Language Standard: ISO C++17 / C99
- Graphics Library: Raylib (Hardware accelerated 2D rendering)
- IDE: Microsoft Visual Studio 2022
- Version Control: Git / GitHub

2.2 Core Engine Design

The application utilizes a modular architecture to ensure separation of concerns. The codebase is divided into the following subsystems:

- Game Loop Manager: Orchestrates the Update() and Draw() cycles. It implements a delta-time (dt) calculation to ensure frame-rate independent gameplay physics, targeted at a stable 60 FPS.
- Entity Manager: Handles the instantiation, updating, and deallocation of dynamic objects (Projectiles, Enemy Units).
- Wave Manager: A state machine that controls the timing, density, and difficulty scaling of enemy spawns.

2.3 Collision & Physics

To optimize performance ($O(n)$ complexity), the system uses simplified geometric checks:

- AABB (Axis-Aligned Bounding Box): Utilized for UI interactions (Mouse vs. Button).
- Euclidean Distance Checks: Utilized for range detection (Tower vs. Enemy) and impact detection (Projectile vs. Enemy).

3. GAMEPLAY SPECIFICATIONS

3.1 Defense Units (Towers)

The system implements three distinct tower classes inheriting from a base Tower struct.

[TYPE: Archer Tower]

- Key: 1
- Role: Single Target DPS
- Technical Behavior: High frequency Update() cycle for projectiles. Low damage per hit.

[TYPE: Melee Tower]

- Key: 2
- Role: Area Denial
- Technical Behavior: Checks collision within a short radius. High damage output.

[TYPE: Ice Tower]

- Key: 3
- Role: Crowd Control
- Technical Behavior: Applies a speed_modifier (< 1.0) to enemy entities within range.

3.2 Enemy Entities

Enemy behavior is governed by a pathfinding algorithm that navigates a vector array defining the road.

- Orc: High Speed / Low HP. Swarm tactic; meant to overwhelm fire rate limits.
- Uruk: Balanced stats. Standard infantry unit.
- Commander: High HP / High Resistance. Elite unit; prioritizes durability over speed.
- Troll: Tank (Massive HP). Slow velocity; acts as a damage sponge for other units.
- Grond (Battering Ram): Special Tank (Very Slow, Huge HP). If it reaches it will oneshot the castle special unit.
- Witch King: Boss unit. Spawns in final waves. Highest HP value in the registry.

3.3 Active Abilities (Player Skills)

During the game blood is accumulated as enemy units are killed. This blood resource can then be spent to cast special spells, such as summoning Gandalf or calling the Rohirrim.

Function: Cast_Gandalf() (Key: Q)

- Logic: Iterates through the active enemy list and sets velocity = 0 for a defined duration 't'.

Function: Cast_Rohirrim() (Key: W)

- Logic: Instantiates a linear damage hitbox that traverses the screen vector, eliminating intersecting entities.

3.4 Resource Management

- **Gold:** The primary currency used to purchase and place towers. Earned by defeating enemies.
- **Uruk Blood (Special Mechanic):** A secondary "mana" resource system.
 - **Accumulation:** "Uruk Blood" is harvested automatically when specific enemy units (particularly Uruks and Orcs) are slain on the battlefield.
 - **Usage:** This resource is required to cast powerful active abilities (Gandalf and Rohirrim) that can turn the tide of battle when towers alone are insufficient.
 - **UI Representation:** Represented by a resource bar distinct from the Gold counter, forcing players to balance tower kills with ability usage.

4. PROGRESSION & GOALS

Difficulty Levels and Bosses

The game features a tiered difficulty system. Each difficulty level introduces unique challenges and culminates in a specific Boss Wave.

- **Difficulty 1 (The Siege Begins):**
 - Focus: Introduction to mechanics and basic enemy types (Orcs/Uruks).
- **Difficulty 2 (The Onslaught):**
 - Focus: Increased enemy density and introduction of armored units.
 - Final Boss: The Commander. An elite unit with high resistance that prioritizes durability. While normally an elite unit, in this difficulty tier, he serves as the primary threat.
 - Final Boss: Battering Ram (Grond). A massive mechanical unit with high health that ignores crowd control effects.
- **Difficulty 3 (Return of the King):**
 - Focus: Maximum wave density, requiring perfect use of the "Uruk Blood" mechanic.
 - Final Boss: Witch King. The ultimate threat. Spawns in the final waves with the highest HP value in the registry.

5. CONTROLS & USER INTERFACE

The input handling system polls hardware state every frame.

- LMB (Left Click): Place Tower / Interact with UI buttons.
- Keys 1-2-3: Select Tower Blueprints (Archer, Melee, Ice).
- Key Q: Cast Gandalf Ability:Stuns every enemy for a while. (consumes Uruk Blood).
- Key W: Cast Rohirrim Ability:Deals damage to all enemies on path.(consumes Uruk Blood).
- Esc: Quit the game.

6. ASSET ATTRIBUTION & LICENSING

All assets utilized in the production of this software are royalty-free.

6.1 Audio Resources

- [Freesound.org](https://freesound.org)
- [MyInstants.com](https://myinstants.com)
- [Voicy.network](https://voicy.network)

6.2 Visual Resources

- Character Sprites: [OpenGameArt.org](https://opengameart.org), [Itch.io](https://itch.io), [Microsoft Bing](https://www.microsoft.com/bing)
- Environment/Structures: [Kenney.nl](https://kenney.nl), [Microsoft Bing](https://www.microsoft.com/bing)
- End-Game Art: [OpenArt.ai](https://openart.ai)

6.3 Typography

- Primary Typeface: Sourced via [DaFont](https://dafont.com).

7. DESIGN CHALLENGES & SOLUTIONS

Challenge: Asset Consistency in Isometric Perspective

- **Problem:** Creating 2D pixel art that mimics a 3D isometric perspective was difficult. Initial AI generations for units like the "Battering Ram" and "Witch King" were often side-scrolling (flat 2D) or top-down, which did not match the isometric grid of the map.
- **Solution:** We utilized iterative prompting with Bing Image Creator (DALL-E 3). By specifically refining prompts to include terms like "isometric view," "game asset," and "high angle," we were able to generate assets that fit the specific perspective. These were then manually cleaned (background removal) and resized to fit the game's collision boxes.

Challenge: Manual Memory Management

- **Problem:** Eschewing engines like Unity meant we had to handle memory manually. Improper deallocation of projectiles or dead enemies led to memory leaks.
- **Solution:** We implemented a modular Entity Manager. This system handles the instantiation and strict deallocation of dynamic objects, ensuring that once an entity (like a projectile) leaves the screen or hits a target, its memory is immediately freed.