# Project Proposal: *No Man's Land*

A Top-Down WWI Horde Survival Game on the DE1-SoC

**Team:** Rohit Biswas (rb3908), Kambinachi Obioha (kno2117), Nicola Paparella (np2953)
**Instructor:** Prof. Stephen Edwards | **Date:** March 14, 2026

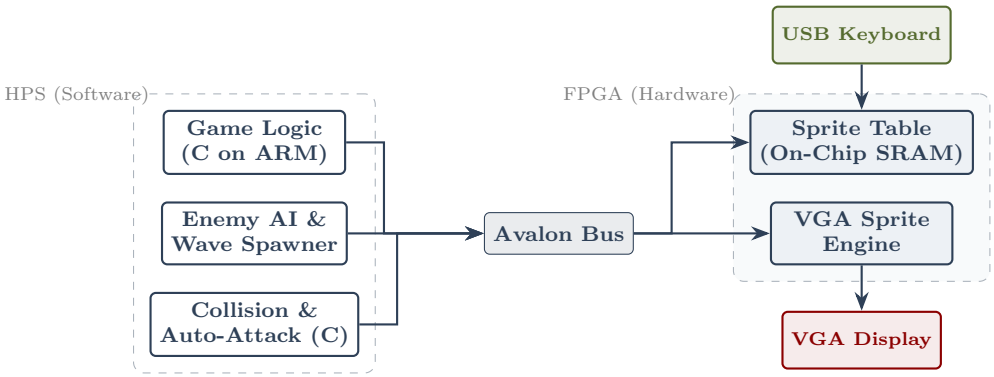
## Project Overview

*No Man's Land* is a top-down horde survival game set in World War I, built on the DE1-SoC FPGA platform. The player controls a soldier, manually aiming and firing at waves of enemies while collecting items that grant autonomous auto-attacking abilities (mortars, barbed wire, mustard gas, artillery barrages). A custom VGA graphics pipeline in SystemVerilog handles sprite rendering and tile-based background display on the FPGA fabric, while game logic, enemy AI, collision detection, auto-attack management, and wave spawning run in C on the ARM Cortex-A9. Player input comes from a USB keyboard (WASD movement, arrow keys for aiming/firing).

Beyond its technical goals, the game uses the horde survival format to deliver a narrative statement about mechanized violence. The enemy composition shifts gradually across waves, from armed combatants to unarmed figures, without announcement. The auto-attack items the player has built up do not distinguish between targets. The end screen displays a kill breakdown by category, recontextualizing the player's performance. The better they played, the more devastating the summary.


Figure 1: Mid-game mockup (Wave 7). Green = player; dark gray = armed enemies; tan = non-combatants. Auto-attacks: barbed wire, gas cloud, mortar.

## System Architecture



### Hardware (SystemVerilog)

**VGA Sprite Engine:** Drives a 640×480 display at 60 Hz. Reads from a sprite table in on-chip SRAM (sprite ID, position, size, active flag), performs per-scanline bounding box checks, and fetches pixel colors from sprite ROM. A tile-based background layer renders the battlefield underneath. Supports up to 32 simultaneous sprites with priority-based overlap.

**Sprite Table (On-Chip SRAM):** Memory-mapped sprite entries written by software and read by the VGA engine each frame. Double-buffered with a frame-sync handshake to prevent tearing.

### Software (C on ARM)

The game state manager runs the main loop: updating positions, resolving collisions, managing score, XP, and wave progression. **Collision detection** runs entirely in software using AABB checks across all active entities each frame; at 32 entities, pairwise checks complete in microseconds on the ARM core. **Auto-attack logic** is also software-driven: on cooldown expiry, the C code computes projectile positions from current game state and writes new entries directly to the sprite table. By design, auto-attacks target the nearest entity without type filtering.

Enemy AI uses a simple chase behavior (move toward player). The wave spawner controls composition, difficulty scaling, and the gradual introduction of non-combatant entities via data-driven wave tables. The item system presents upgrade choices on level-up and configures auto-attack parameters in software. A USB HID driver maps keyboard input to movement and aim/fire commands.

### HW/SW Communication

Memory-mapped registers on the Avalon lightweight bus: sprite table (SW→HW), player state (SW→HW), and keyboard input (HW→SW). Software writes sprite positions, active flags, and sprite IDs; hardware reads them during scanline traversal.

## Milestones and Division of Labor

| Week | Milestone | Owner(s) |
|---|---|---|
| 1–2 | VGA tile rendering; USB keyboard input; basic sprite display | All |
| 3–4 | Player movement and shooting; enemy spawning and chase AI; SW collision | Rohit, Kambi |
| 5 | Auto-attacks from SW; item selection UI; narrative wave progression | Nicola, Rohit |
| 6 | Integration testing, visual polish, bug fixes, demo preparation | All |

| Member | Primary Responsibilities |
|---|---|
| Rohit | Game logic/state management (C); USB driver; Avalon register design; integration |
| Kambi | VGA sprite engine and tile renderer (SV); sprite ROM; VGA timing pipeline |
| Nicola | Auto-attack logic (C); enemy AI and wave spawning; item/upgrade system |

## Technical Risks and Mitigations

| Risk | Mitigation |
|---|---|
| Sprite count exceeds 60 fps rendering bandwidth | Hard cap at 32 sprites with priority culling; degrade gracefully |
| USB keyboard drops simultaneous keypresses | Early N-key rollover testing; higher polling frequency fallback |
| HW/SW sync issues (tearing, stale data) | Double-buffered sprite table with frame-sync handshake |
| Software game loop too slow for 60 fps update | Profile early; simplify AI or reduce entity count as needed |