Sacramento State University

Spring 2025

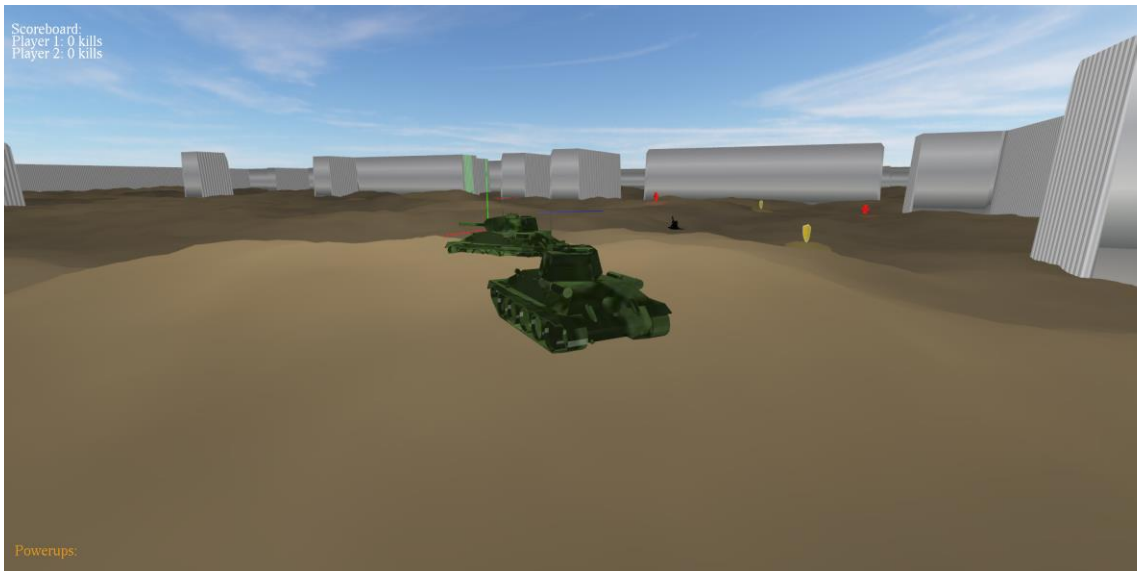
Big Boom

Player Guide

Nicholas Lewis & Tyler Burguillos
CSC 165-02
Dr. Scott Gordon

Table of Contents

2 – Game Screenshots



3 – How to Compile and Launch the Game

This section guides you through the full setup process needed to build and launch the game.

3.1 – Installing and Building TAGE

To run this game, you'll need to first install and build the TAGE engine, which provides the rendering, input, and audio systems used in the project.

1. Download & Install Java JDK version 17
    - This is available on the Oracle website. You need the JDK, not just the JRE.
    - After installing, look for where Oracle put it. Usually that is `C:\Program Files\Java\jdk-17`
2. Modify your PATH environment variable
    - Go into the Control Panel > System > Advanced System Settings > Environment Variables.

- Under *System Variables*, open the PATH variable. There is likely to be an entry that reads `C:\Program Files\Common Files\Oracle\Java\javapath`
- Replace that entry with Java's "bin" folder. Usually, that is `C:\Program Files\Java\jdk-17\bin`

3. Create the JAVAGAMING folder to hold OpenGL-related libraries
    - Preferably `C:\javagaming` Create this folder.
4. Copy the JAVAGAMING libraries from the included `tage_build.zip` file.
    - Copy those five folders into the `C:\javagaming` folder on your machine
5. Add the JINPUT library to your PATH environment variable
    - Go back into the PATH environment variable (as you did earlier in step #2).
    - Add the following entry: `C:\javagaming\jinput\lib`
6. Create a CLASSPATH variable (if you don't already have one)
    - Go back into the Control Panel > System > Advanced System Settings > Environment Variables.
    - Under *System Variables*, see if there is a variable named CLASSPATH.
    - If there isn't, then create one by clicking the "New…" button, then in the "name" field, type in CLASSPATH. In the "value" field, type in a period (.") by itself.
7. Add JOGL, JOML, JINPUT, JBULLET, and VECMATH to the CLASSPATH variable
    - While still in the Environment Variables, open up the CLASSPATH variable by double-clicking it. You may see a table entry tool, or you may see the previous dialog box from step 6(a). Add entries for the five libraries .jar files, after the single "." entry. These should be *full paths.*
    - The entries would look like this:
        - `C:\javagaming\jogl\jogamp-fat/jar`
        - `C:\javagaming\joml\joml-1.10.7.jar`
        - `C:\javagaming\jinput\jinput.jar`
        - `C:\javagaming\jbullet\jbullet.jar`
        - `C:\javagaming\vecmath\vecmath.jar`
    - `If you are entering them in the single line dialog box, separate them with semi-colons, like this:`
    - `C:\javagaming\jogl\jogamp-fat.jar;C:\javagaming\joml\joml-1.10.7.jar` etc.

`Note: changes to environment variables don't take effect until rebooting. (alternatively, you can open a new command window and work from there)`

3.2 – Building the TAGE Class Files

It is best practice if you have been testing multiple games which have been built using the TAGE engine to make sure any old class files are deleted from your system and run a fresh build for each project to ensure that any changes a team made to the TAGE engine are reflected properly and to avoid potential crashes if people are using the same method names with different constructors.

1. Open a new command window and migrate to the “BigBoom” folder.
2. Type “clearTAGEclassFiles”. You should see a series of messages as old TAGE class files are cleared in preparation for a new build. If you have never built TAGE class files before, it is normal for some warning messages to also appear.
3. Type “buildTAGE:. You should see a series of messages as each TAGE component compiles. If the libraries were installed correctly, there shouldn’t be any messages or warnings.

To streamline the development process, we provide two batch files: **`compile.bat`** and **`run.bat`**, located in the root directory of the project. These files handle all compilation and launch procedures for both the client and server codebases.

Compile Script (`compile.bat`)

The `compile.bat` script compiles both the **Client** and **Server** Java files while logging any compilation errors to separate .txt files:

- `compile_client_errors_.txt` for client compilation issues
- `compile_server_errors_.txt` for server compilation issues

If errors occur, they are printed to the terminal. Otherwise, a success message confirms that compilation completed without issues.

Run Script (`run.bat`)

The `run.bat` script prompts the user to choose which mode to launch:

1. **Single-Player Mode (non-networked):**
    - Automatically fetches your local IP address using `ipconfig` and launches the game in single-player mode.
    - Let's you select between **FAST** and **SLOW** Tank
2. **Server Mode**:
    - Launches the server. This must be run in a separate terminal before launching clients.
3. **Client Mode:**

- Prompts the user to enter a server IP address, then launches the game in client mode.
- Let's you select between **FAST** and **SLOW** Tank

NOTE: Before launching any multiplayer clients, you **MUST** start the server by selecting **Option 2** in a separate terminal window. For convenience, the person hosting the server can use **Option 1 (Single-Player Mode)** to connect to their own server instance, since we've designed the script to automatically detect and use their local IP address.

4 – Objective and Gameplay Mechanics

Overview:

Big Boom is a fast-paced tank battle game set in a handcrafted maze defined by a secondary heightmap layered over natural terrain. Players navigate the maze, eliminate opponents, and survive long enough to outscore the competition – all while avoiding deadly turret fire and collecting strategic power-ups.

Goal:

The objective is to earn the highest score by eliminating enemy tanks, avoiding AI turrets, and collecting power-ups. The first player to get 5 kills wins.

Win Conditions (Configurable):

- **First to N Kills:** The match ends when a player reaches a target kill count (5).

Lives & Respawning:

- Upon death, layer health resets and the kill is added to the scoreboard.
- AI turrets do **not** respawn and can only be disabled if a player moves far enough away from it.

Scoring System:

- number of Kills player receives

AI Turrets:

Turrets guard strategic chokepoints. Each turret has three animation states:

- **Activate:** Lowers from idle position
- **Scan:** Sweeps horizontally when a player reaches a certain range.

- **Deactivate:** Raises back to idle when no players are nearby.
- Note: These animations do not work on school lab machines and only worked on our personal machines.

Turrets transition smoothly between states and briefly delay targeting after animation transitions to ensure realistic behavior. Once a player is within firing range, the turret will interrupt its animation and begin firing at the nearest player.

Power-ups:

Power-ups spawn in randomized valid locations within the maze. Each one disappears after being collected and respawns after a cooldown.

- **Speed Boost:** Temporarily increased movement speed by a considerable amount.
- **Shield:** Temporarily makes the player invulnerable to damage for a short period of time.
- **Health Regen:** Increases health by 20 points (if not at max).

HUD Elements:

- Scoreboard
- Power-up status

Multiplayer Notes:

Power-up states and turret AI are synchronized across all clients through the server. This ensures fairness and consistent behavior regardless of client-side performance.

5 – Control Scheme (Keyboard | Controller)

Move Forward	W
Move Backward	S
Turn Left	A
Turn Right	D
Turn Turret Left	Left Arrow
Turn Turret Right	Right Arrow

Tilt Gun Up	Up Arrow	
Tilt Gun Down	Down Arrow	
Shoot Gun	Space Bar	
Show Health Bar	H	
Turn Lights On	E	
Move Camera Left		Z-Axis (Right Stick)
Move Camera Right		Z-Axis (Right Stick)
Move Camera Up		RZ-Axis (Right Stick)
Move Camera Down		RZ-Axis (Right Stick)
Zoom Camera In		Y-Axis (Left Stick)
Zoom Camera Out		Y-Axis (Left Stick)
Free Flight Mode (testing)	T	
Take Damage Mode (testing)	K	
Toggle physics simulation (testing)	P	
Toggle tank headlight	F	

6 –Lighting Use in Game

The game features a combination of ambient and spotlight lighting. **Ambient** light provides overall scene visibility, while spotlights enhance realism and gameplay. Tanks have functional **headlights** that illuminate their path, and each power-up is marked with a distinct colored **spotlight** (green for speed, red for health, yellow for shield). Players can press E to toggle a dark mode, which dims the ambient lighting and increases reliance on localized light sources for navigation and immersion.

7 – Network Protocol Changes

Our custom UDP-based protocol was expanded significantly to support new features like tank selection, synchronized turret and gun rotation, health updates, scoreboards, and power-up state sharing. We added new message types (`gunRot`, `turretRot`, `health`, `scoreboard`, `headlight`, `powerupSync`, and `turretState`) to ensure all clients maintain accurate representations of remote avatars and game objects. The protocol now distinguishes between "fast" and "slow" tank models at join time and keeps all ghost avatars synchronized with precise transformation matrices. These additions were designed to enhance gameplay realism and multiplayer consistency while preserving efficient communication over UDP.

7 – TAGE Changes and Additions

We extended the TAGE engine by adding a new `CameraOrbit3D` class that allows for smooth third-person camera control around the player's tank. This feature supports dynamic orbiting via gamepad input, including zoom, elevation, and azimuth control, and accounts for the turret's yaw to enhance immersion.

We also modified the `HUDmanager` to support multi-line text using `\n` characters, enabling more readable and organized HUDs such as a scrolling scoreboard or event logs. These additions provide greater camera flexibility and more dynamic on-screen information for players.

8 – Game Overview Statement

1. **Genre**:
   Multiplayer Third Person Shooter
2. **Theme**:
   Tactical futuristic tank warfare in a physics-based environment, where players compete in a hostile terrain featuring AI turrets, and power-ups.
3. **Dimensionality**:
   3D (Three-Dimensional)
   The game features fully modeled and animated 3D tanks, terrain, and environmental objects. Players navigate and interact in a real-time 3D world with camera orbit control and spatial audio.
4. **Activities Utilized**:
   a. Navigating a terrain-based map using a tank avatar
   b. Engaging in combat with other players and AI turrets (Couldn't get AI turrets to shoot ran out of time)
   c. Collecting power-ups (speed boosts, health boosts, shields)

d. Managing resources like health and visibility (e.g., headlights and dark mode)
e. Using HUDs to monitor game status, including scoreboards and power-up effects
f. Multiplayer interaction and synchronization through a client-server architecture with ghost avatars
g. Dynamic turret and gun aiming with camera and rotation synchronization

9 – Requirement Visibility Breakdown (In-Game)

External Models:
The game includes multiple custom-made OBJ models created in Blender, such as the fast tank, slow tank, and the power-up objects (speed boost, health boost, and shield). These models are clearly visible during gameplay. Each team member contributed at least one model.

Custom UV Textures:
Each custom model has a unique UV-unwrapped texture. For example, the fast tank uses `fastTank.png` and the slow tank uses `slowTank.png`, both of which are visibly mapped to their respective tank bodies. The textures are designed specifically for the models and are not random or tiled patterns.

Networked Multiplayer:
The game supports two-player multiplayer over the network. When both players join, they can see each other's tank avatars in real time. Each player's turret and gun rotate, headlights toggle, and health changes are visible on the ghost avatar of the other player.

Character Selection:
At game startup, players choose between a fast or slow tank by passing a command-line argument. The selected tank model is visible in the game and other clients see the correct model thanks to networked CREATE messages.

Skybox:
The game world is surrounded by a visible cube map skybox ("battlefield") that gives the environment a full 360° background.

Terrain:
The tank drives on a height mapped terrain generated from a texture (`terrain_height.png`). Players can clearly see hills and elevation changes as the tank moves.

Lighting:

At least three lights are used in-game. A headlight is attached to each player's tank and can be toggled with the 'F' key. Colored spotlights hover above each power-up: green for speed boost, red for health, and yellow for shield. There is also ambient global lighting that can be toggled with 'E' to switch to dark mode.

HUD:

A HUD is shown in the top-left with currently held power-ups, such as "Speed Boost" or "Healed". A scoreboard appears in the top-right listing kill counts for each player. The health bar display can also be toggled with the 'H' key.

3D Sound:

There are at least two 3D sounds: a proximity-based turret/mine beep that plays louder as you get closer to the turret, and ambient background music that fades gradually. These sounds spatially change based on the player's position and add immersion.

Hierarchical SceneGraph:

The tank model is structured hierarchically using TAGE's scenegraph. The tank body is the root, with the turret as a child, and the gun as a child of the turret. This hierarchy ensures the turret and gun follow the tank's movement while rotating independently.

Animation:

Three different animations for AI turret which are de-active, active, scanning. They all use skeletal animation made in.

NPC (AI-Controlled):

A turret is placed in the environment that rotates to face the closest player. It is controlled by an AI controller, not random behavior. The AI logic rotates the turret to track the player tank's position, making it a dynamic non-player character.

Physics (JBullet):

Physics is fully integrated using TAGE's JBullet engine. The tank uses physics to shoot at powerups and enemy tanks. Power-ups are also physics objects, and collisions with the player trigger effects like boosting speed or healing. You can toggle the visual physics debugger with the 'P' key.

10 – Requirements Not Working

All requirements should be met. The game was not as finished as we like.

12 – Techniques Going Beyond What Was Required

- **Advanced Orbit Camera with Turret Tracking**
  We implemented a dynamic third-person orbit camera system (`CameraOrbit3D`) that tracks not only the avatar but also adjusts based on the tank turret's yaw. This creates a more immersive and responsive camera experience.
- **Turret AI with Player Targeting**
  Our turret object automatically detects and rotates toward the nearest player in real time. This uses vector math and `atan2` to calculate and apply smooth turret rotation.
- **Player Headlight with Network Sync**
  We added a spotlight headlight to each player's tank that can be toggled with a key. The on/off state is synced across the network, so other players see it update correctly.
- **Distance-Based Manual Sound Fading**
  Instead of relying only on JOAL's built-in roll off, we implemented a custom distance-based volume fade for the turret's beeping, giving us more control over how it sounds at different ranges.
- **Health Bar with Live Scaling & HUD Feedback**
  The player's health bar is a 3D object that scales in size based on current health. It follows the avatar and updates in real time. The HUD also reflects when the player picks up speed, health, or shield boosts.
- **Custom Power-up Lighting System**
  Every power-up has a spotlight above it with color-coded lighting (green for speed, red for health, yellow for shield). These lights follow their associated power-up and enhance visual feedback.
- **Switchable Tank Models**
  Players can choose between a fast and a slow tank at startup via command-line argument. This selection is used to load different models, textures, and scaling behavior.
- **Optional Dark Mode Toggle**
  Pressing a key enables "dark mode" by lowering global ambient light, adding a visual challenge and atmosphere not required by the spec.
- **Terrain-Following and Free-Fly Toggle**
  We implemented a feature to toggle between terrain-following movement and free-flying mode, giving players two distinct movement styles.

13 – Contributions of Each Team Member

Nicholas Lewis:

- Designed the **shield power-up model** using Blender and applied a custom UV-mapped texture.
- Implemented the **networking system**, including:
  - Multiplayer support using UDP/TCP.
  - Ghost avatar creation and updates.
  - Protocol extensions to sync turret/gun rotation and headlight state.
- Added all **lighting systems**, including:
  - Headlights on tanks (toggleable).
  - Spotlights above power-ups.
  - Global ambient and positional lighting.
- Added all **3D sounds**:
  - Turret beeping with manual distance-based fading.
  - Background ambient audio with smooth fade-in.
- Created and managed the **HUD system**, including:
  - Health bar rendering and scaling.
  - Text-based HUD for power-up effects and scoreboard.
- Integrated the health system:
  - Damage handling.
  - HUD alerts for shield, health boost, and speed boost.

Tyler Burgillos:

- Designed the **speed boost** and **health boost** models in Blender, with custom UV-mapped textures.
- Created the **hierarchical tank structure**:
  - Split the tank into separate **bodies**, **turret**, and **gun** objects.
  - Allowed turret and gun to rotate independently.
- Implemented **terrain and maze** using heightmaps with collision physics.
- Added **skybox** with dynamic visuals.
- Built the **physics system** for tank and power-up interactions using JBullet.
- Created the **power-up system** with activation logic and timers.
- Collaborated with Nicholas on the **AI turret system**, which tracks the closest player and rotates toward them in real time.

14 – Assests We Created

Assets We Created Ourselves:

- **Shield model** — designed and UV-textured by Nicholas in Blender.
- **Speed Boost model** — designed and UV-textured by Tyler in Blender.
- **Health Boost model** — designed and UV-textured by Tyler in Blender.
- **All power-up textures** — custom-created to match their respective models.

Assets Provided by CSc-155 or CSc-165:

- **Axis line shapes** (X, Y, Z).
- **Dolphin model** — included in early course labs (not used in our game).

All Other Assets:

- All other models, textures, heightmaps, sounds, and related assets were obtained from **external sources** outside of CSC-155 and CSC-165.

15 – Assests not listed in #14

For all assets not created by our team or provided by CSc-155/CSc-165, we ensured they were obtained from free, license-safe sources appropriate for academic use:

- **Tank Model and Turret**
  Source: Free3D.com
  License: Free for non-commercial and educational use. Attribution not required but encouraged.
  Models were downloaded and then split into `tankBody.obj`, `tankTurret.obj`, and `tankGun.obj` by Tyler.
- **Skybox**
  Source: Terra-Gen – A Website for Terra-Gen LLC.
- **Terrain and Maze Heightmaps**
  Source: CPetry's Texture Generator
  License: Free online tool that allows generation and export of custom procedural textures and heightmaps for personal or educational use. No attribution required.
- **Tank Textures**
  Source: Textures.com

License: Free account allows up to 15 texture downloads per day for non-commercial use, including educational projects. Attribution optional but recommended.

- **Sound Effects** (`minebeeping.wav`, ambient background)
  Source: FreeSound.org
  License: All sounds used were labeled as **CC0** (public domain) or **CC-BY**, and we ensured compliance by choosing assets with no commercial or attribution restrictions for academic use.

We avoided any copyrighted, paid, or attribution-restricted content and only used assets marked as free for **non-commercial and educational purposes**, ensuring our use complies with licensing guidelines. Screenshots and links to source pages can be provided upon request.

16 – Machine Tested On

ECS- SPACEQUEST, ECS-CRASH, ECS-ASTEROID