Antillery

A 2D Artillery-Style Game

CPSC 490 – Capstone Project Proposal

Group Member	CWID
Josh Lollis	887031649

1 Table of Contents

2 Abstract

This capstone proposal outlines the design and development plan for Antillery, a 2D turn-based artillery game blending classical destructive mechanics with peace-oriented tools.

The initial sections define Antillery's concept, historical influences, and capstone justification. The proposal then identifies how the game introduces strategic tension through pacifist mechanics, outlines development objectives, and presents the project's feasibility, scope (both in and out), and unique contribution to the artillery genre.

The following sections define the target audiences, vision statements, and elevator pitches for multiple stakeholder types. The document also outlines when and how development will occur, including the project timeline, key tasks, and technical tools in use. It concludes with anticipated risks, mitigation strategies, expected deliverables, and the planned scope beyond the CPSC 491 term.

3 Introduction

This proposal outlines the concept, inspiration, and creative goals behind *Antillery*, a senior capstone project that aims to turn the artillery game genre on its warhead. However, before venturing into the goals and technical aspects of the project, we should cover some introductory details, including a broad view of the historical context and concept of *Antillery*.

3.1 Project Overview

Antillery is a satirical take on the beloved artillery game format. Inspired by the iconic "*Worms*" franchise created by Team17, *Antillery* maintains the same strategic, turn-based, destructively chaotic gameplay typical of an artillery-type game while introducing peace-oriented mechanics that border on disruptive and destructive in their own right. In addition to the traditional arsenal of artillery, *Antillery* arms players with a new category of "eco-friendly" and "nonviolent" weaponry and tools–eco-friendly, restorative, or passive-aggressive, but still disruptive and impactful in battle.

3.2 Historical and Conceptual Context

3.2.1 Disclaimer: Source Material Limitations

Because artillery games occupy a relatively narrow space in gaming history, academic sources on the genre are limited. Much of the historical content regarding early gaming culture comes from enthusiast communities rather than formally rigorous scholarly sources.

This paper relies on "Scorched Parabolas: A History of the Artillery Game" by Dr. Matt Barton, a respected gaming historian whose work balances scholarly depth and subject matter familiarity. Originally published in Volume 4 of Armchair Arcade, Barton's article offers a grounded look at the genre's growth, from early mainframe simulations to later commercially successful franchises (Barton).

Though not a peer-reviewed academic source, Barton's writing is sufficiently rigorous for the needs of the introductory portion of this paper (Barton).

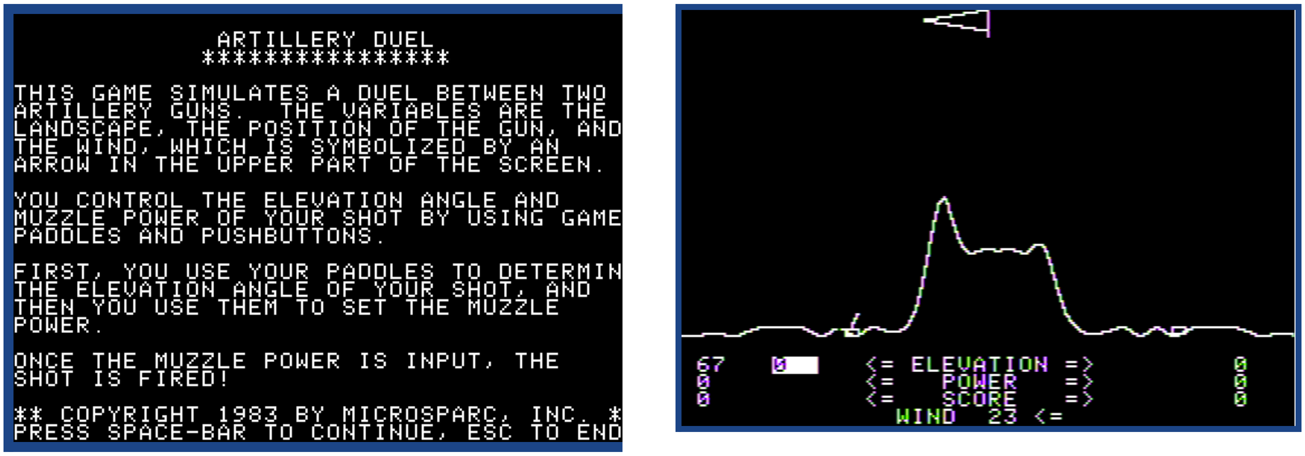
3.2.2 A Brief History of Artillery Games

3.2.2.1 Text & Vector-Based Simulation: Origin Story of Artillery "Games"

The earliest artillery games emerged as text-based simulations on mainframe computers in the 1960s. While primitive compared to today's games, these early experiments in electronic entertainment established many of the core mechanics defining the genre for decades (Barton).

The earliest simulations were developed using BASIC and were entirely text-based, requiring players to imagine the evolving battlefield. Players took turns entering parameters such as barrel angle and firing velocity. After processing the simulation, the mainframe would print the results, detailing hits, misses, and the conditions of the battlefield. These early simulations had a limited player base of mainframe operators who implemented and circulated their new features.
Although direct visual records of mainframe-era artillery games are scarce, their design legacy persisted. By the early 1980s, successors like Artillery Duel (1983) began introducing simple vector graphics and visual interfaces on home consoles (MobyGames).

Even with their limited complexity and audience, these early simulations established the core mechanics that later artillery games would develop further.



```
           ARTILLERY DUEL
          *****************

THIS GAME SIMULATES A DUEL BETWEEN TWO
ARTILLERY GUNS.  THE VARIABLES ARE THE
LANDSCAPE, THE POSITION OF THE GUN, AND
THE WIND, WHICH IS SYMBOLIZED BY AN
ARROW IN THE UPPER PART OF THE SCREEN.

YOU CONTROL THE ELEVATION ANGLE AND
MUZZLE POWER OF YOUR SHOT BY USING GAME
PADDLES AND PUSHBUTTONS.

FIRST, YOU USE YOUR PADDLES TO DETERMIN
THE ELEVATION ANGLE OF YOUR SHOT, AND
THEN YOU USE THEM TO SET THE MUZZLE
POWER.

ONCE THE MUZZLE POWER IS INPUT, THE
SHOT IS FIRED!

** COPYRIGHT 1983 BY MICROSPARC, INC. *
PRESS SPACE-BAR TO CONTINUE, ESC TO END
```

```
67   0     <= ELEVATION =>          0
0          <=   POWER   =>          0
0          <=   SCORE   =>          0
             WIND  23 <=
```

Figure 3.2.2.1. *Artillery Duel* (1983) (MobyGames).

3.2.2.2 *Scorched Earth*: Evolving Artillery Through Endless Customization

Scorched Earth was released in 1991 and marked a pivotal moment, breathing new life into the artillery genre (Barton). Unlike its text-based predecessors, it was in full color, real-time, and standardized many of the hallmark conventions of artillery games, much like what Super Mario Bros. did for platformers (Barton).

Scorched Earth expanded significantly on the foundational mechanics of earlier artillery games by implementing over thirty weapons, ranging from conventional missiles to napalm. Much of its acclaim came from its level of strategic depth, afforded by its fully destructible terrain, customizable parameters such as wind and gravity, and a robust in-game economy.

This landmark artillery title introduced several design ideas that became hallmarks of the genre today. Its focus on dynamic strategy through choice and customization shaped the landscape of future artillery games. These principles influence some of the design objectives of Antilery.

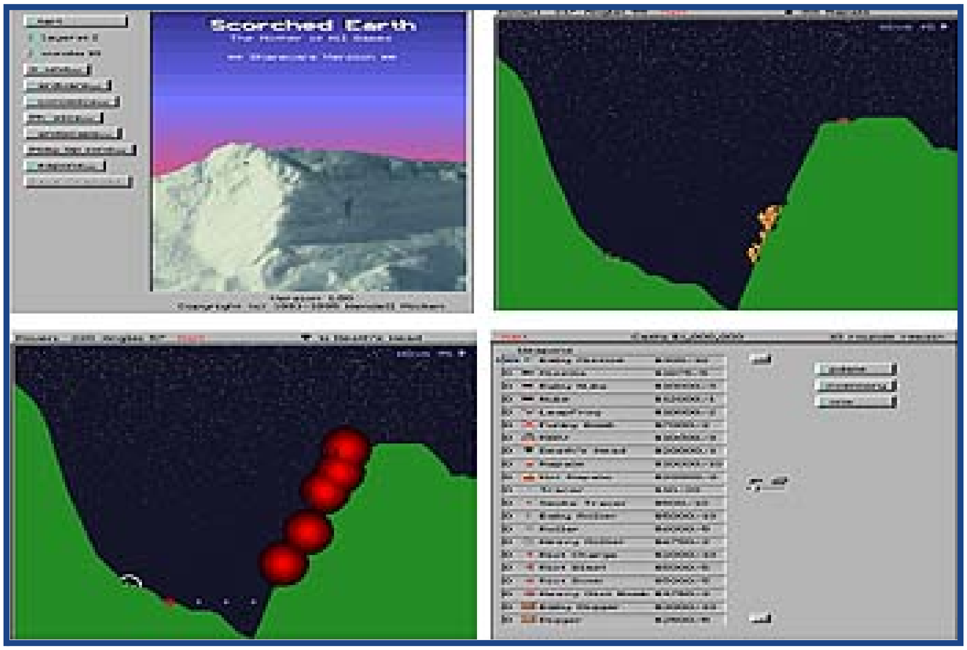


Figure 3.2.2.2. Scorched Earth (1991) UI and terrain interaction (Barton).

3.2.2.3 *Worms*: Giving the Artillery Genre a New Backbone

Worms, released by Team17 in 1995, marked a turning point in the artillery game genre. While its gameplay preserved core projectile mechanics, its presentation, a squad of cartoon worms wielding bazookas, shifted the genre from simulation to satirical multiplayer chaos.

Creator Andy Davidson entered the game, called initially Total Wormage, into a competition by Amiga Format magazine. Though it did not win, it caught Team17's attention, which went on to publish it across multiple platforms (Nuttall 26; Barton). Worms emphasized player mobility, absurdity, and personality, unlike its tank-centric predecessors. Worms could walk, jump, and swing across terrain using ninja ropes and parachutes, adding momentum and physical comedy to a genre previously rooted in physics realism. Barton observed, "The worms can do much more than tanks," describing moments of dramatic acrobatics (Barton).

Its tone embraced British comedic absurdity, transforming once-serious combat into a stage for slapstick and satire. Exploding sheep, banana bombs, and absurd weapons blurred the line between threat and punchline. Cheeky voice lines like "First blood!" and "This worm is an ex-worm!" gave the game personality and flair, establishing a new kind of identity for artillery games, one dripping in irony, theatricality, and joyful nonsense (Barton).

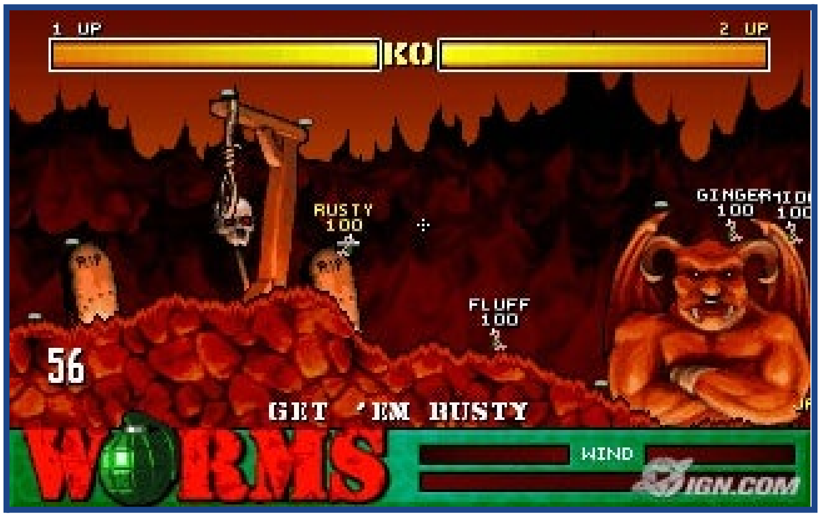


Figure 3.2.2.3. *Worms* (1995) featuring destructible terrain and satirical visuals (LaFlame).

3.3 Antillery Comes Marching In

With this historical and conceptual background in place, the following sections transition from analysis to proposal, exploring Antillery's game mechanics, audience considerations, technical development plan, and capstone justification. Each component builds toward a complete implementation strategy for a playable, satirical artillery game by the end of the academic term.

4 Capstone Summary

4.1 Breadth of Knowledge in Computer Science

Antillery applies core Computer Science principles across multiple disciplines, including programming, software architecture, real-time systems, and project management. The project reflects a complete and practical understanding of the undergraduate curriculum.

It is built in Unity using the C# scripting language to implement real-time projectile physics simulation, modular input handling, and scalable game-state management. The development follows week-long Agile sprint cycles tracked with GitHub version control. The codebase is structured with scalable, modular design principles, using patterns such as the State Pattern and Unity's ScriptableObjects to manage complexity and support maintainability.

Antillery meets the breadth of expectations for a Computer Science capstone by combining software engineering, modular system design, and project management. It demonstrates the ability to plan, build, and maintain a scalable project across multiple technical domains, showing the level of complexity expected in the software industry.

4.2 Difficulty Appropriate for C.S. Degree

Antillery presents a difficulty appropriate for a senior Computer Science capstone project. It requires designing, implementing, and coordinating multiple interdependent systems while maintaining stability, modularity, and scalability.

The project implements real-time project physics simulation, modular input handling, and dynamic state management within an interactive environment. Encapsulating each system in the codebase prevents cascading issues throughout the program. This approach also makes debugging and testing more manageable as the project scales. Because Antillery is designed for flexibility and extensibility rather than as a one-off prototype, it demands greater architectural planning and raises the overall technical challenge of development.

Antillery addresses interconnected engineering challenges while maintaining scalable architecture and system encapsulation. The project reflects the problem-solving demands of entry-level developers in the software development industry, making its difficulty appropriate for a Computer Science capstone project.

4.3 Solution to an Unmet Need in the CS Landscape

Modern software products, especially in gaming, increasingly focus on online dependency, often at the cost of player experience and software ownership. As this shift progressed, offline, player-driven experiences became harder to find. It reflects a broader concern within Computer Science: the growing gap between what users want and what products deliver.

Antillery responds by being a self-contained, offline-first game that puts player accessibility, ownership, and local "couch" multiplayer at the core of its design philosophy. Players do not need persistent internet connectivity or extra subscription services to enjoy the experience.

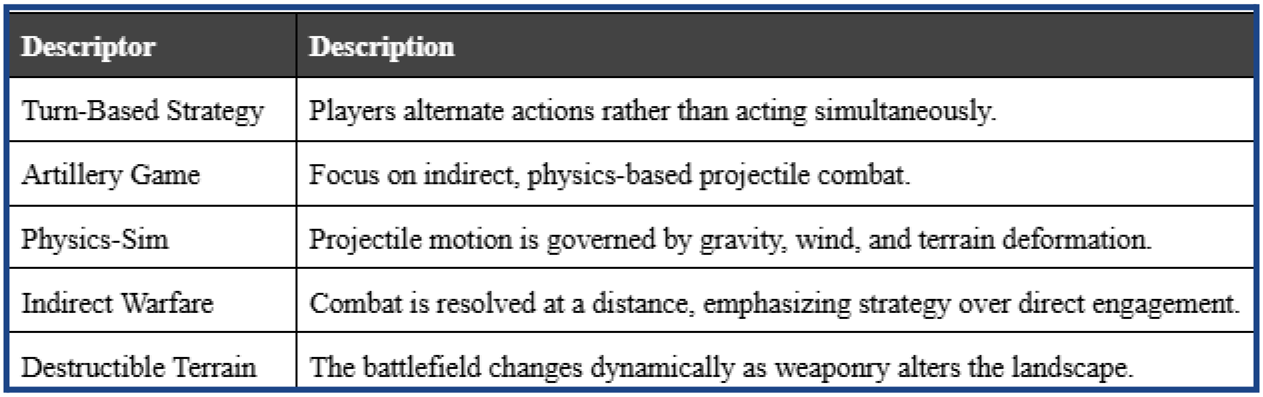
This proposal briefly introduces how Antillery addresses the need for experience-first development. Later sections, including Objectives and Targeted Problems, provide a detailed examination of the user-centered design choices and broader project goals within the Computer Science landscape.

5 What Is Antillery?

As discussed in the Introduction, Antillery draws from the tradition of 2D artillery games, particularly the Worms series. It builds on the genre's core loop of indirect, physics-based combat within a turn-based system while introducing new thematic and mechanical directions. This section examines Antillery's place within the artillery genre, outlining its core gameplay features and thematic goals.

5.1 Genre Classification

Antillery fits the classification of a 2D turn-based strategy game. It belongs to the artillery subgenre, a branch of turn-based strategy centered around indirect, physics-based projectile combat between players. Artillery games typically involve players alternating turns to reposition units, aim and fire projectiles, and adapt to environmental variables such as wind or destructible terrain. These features create emergent gameplay scenarios that are easy for most players to pick up but have a level of strategic depth that offers extensive replayability. **Table 5.1** summarizes commonly used terms for classifying artillery games like Antillery (Barton).



Descriptor	Description
Turn-Based Strategy	Players alternate actions rather than acting simultaneously.
Artillery Game	Focus on indirect, physics-based projectile combat.
Physics-Sim	Projectile motion is governed by gravity, wind, and terrain deformation.
Indirect Warfare	Combat is resolved at a distance, emphasizing strategy over direct engagement.
Destructible Terrain	The battlefield changes dynamically as weaponry alters the landscape.

Table 5.1. Some commonly used terms for describing artillery games.

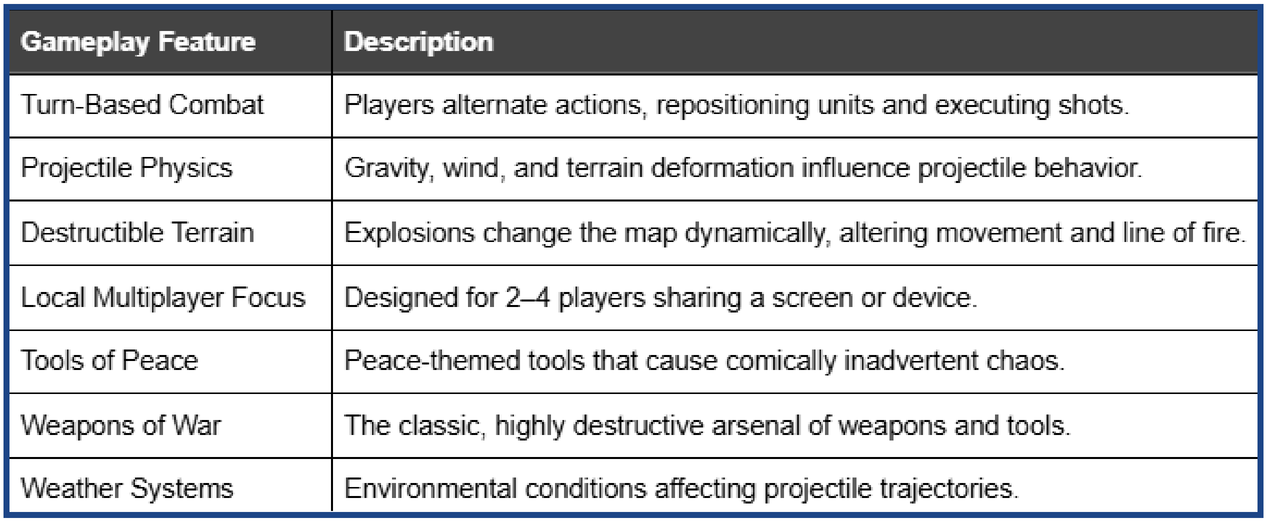
5.2 Core Gameplay Features

5.2.1 Core Turn Structure

Fundamentally, Antillery follows the turn-based style of combat typical of artillery games. Two to four players compete on a 2D battlefield, each taking turns repositioning their active unit, choosing weapons or tools, aiming, and firing projectiles at one another. Gravity and wind affect each shot's trajectory while impacting weapons reshape the destructible terrain throughout the match.

5.2.2 Local Multiplayer and Emergent Chaos

Antillery encourages local multiplayer, with two to four players sharing a single device and screen. This design brings players to the same couch as they commit absurdly large-scale destruction on the battlefield or wage acts of aggressive environmental protest on one another in real time. The game's design leans into this immediacy, turning small actions into big, game-changing, friendship-ruining moments and keeping the competition lively and unscripted. **Table 5.2** summarizes Antillery's core gameplay features.



Gameplay Feature	Description
Turn-Based Combat	Players alternate actions, repositioning units and executing shots.
Projectile Physics	Gravity, wind, and terrain deformation influence projectile behavior.
Destructible Terrain	Explosions change the map dynamically, altering movement and line of fire.
Local Multiplayer Focus	Designed for 2–4 players sharing a screen or device.
Tools of Peace	Peace-themed tools that cause comically inadvertent chaos.
Weapons of War	The classic, highly destructive arsenal of weapons and tools.
Weather Systems	Environmental conditions affecting projectile trajectories.

Table 5.2. Core gameplay features that Antillery includes.

5.3 Inspiration

This section explores how Antillery draws its primary inspiration from the creative breakthroughs introduced by the Worms franchise. As Worms reshaped the artillery genre in its era, Antillery's goal is to be the next evolutionary step in that lineage.

5.3.1 Worms: Innovation of Artillery

The Worms series changed how players thought about artillery games. Instead of focusing only on careful calculations and perfect firing angles, Worms made strategy unpredictable. Humor and player mobility turned each match into something memorable, improvised, and full of surprises.

As discussed in Section 3, Barton highlights how Worms reframed artillery combat through a Monty Python-style comedic lens, replacing grim simulations with colorful, irreverent chaos (Barton). Beyond its humor, Worms gave players absolute freedom of movement. Characters could jump, crawl, and swing across the battlefield, reshaping strategies that were once upon a time reduced to firing arcs (Nuttal 26).

This shift had a lasting impact. The Worms series did not just add jokes; it redefined what an artillery game could be. Its success showed that strategy could come from chaos as easily as calculation. Antillery draws from that innovation but goes further, blending new gameplay mechanics with modern cultural satire to create a new brand of chaotic vision.

5.3.2 Antillery: Evolution of Innovation

Antillery's goal is to push the boundaries of artillery combat, drawing inspiration from Worms. Worms redefined artillery combat through absurdist humor and player mobility. Antillery takes its lead by blending modern cultural satire with new peace-driven gameplay mechanics.

Players still have access to traditional artillery weapons like bazookas and grenades. However, in Antillery, they may also employ new strategies centered on peace, love, and environmental restoration. Nevertheless, even actions of peace, love, and environmentalism sometimes create destructive outcomes. A sunflower might create a shielding canopy, an isolating barrier, or a bridge to strategic ground. In Antillery, every action, whether peaceful or aggressive, becomes part of a larger strategy shaped by the player.

The goal of Antillery is not to apply new paint to a formulaic experience but to evolve the artillery genre in the way that Worms did in its day. In Antillery, players must balance war and peace, challenging them to rethink what combat can be. In doing so, Antillery honors the spirit of innovation introduced by Worms while pushing the artillery genre into a new frontier.

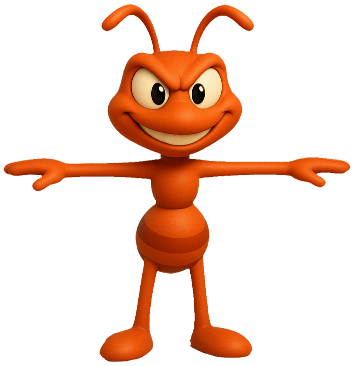


Figure 5.3.2.1. *Red Ant Character Model.*

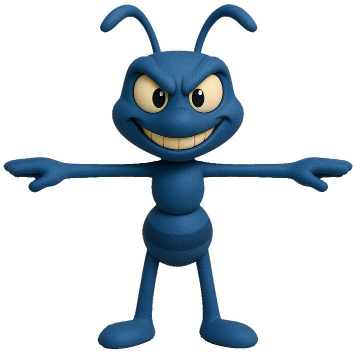


Figure 5.3.2.2. *Blue Ant Character Model.*

6 Why Antillery?

6.1 Reintroducing a Legacy Genre with New Priorities

Antillery is a vehicle to share a timeless and personally significant gameplay experience with a modern audience. It draws from the artillery games of the 1990s, which emphasized local multiplayer, strategic play, and unpredictable outcomes. *Antillery* adapts those classic titles' core structures, employing new themes and mechanics without remaking them.

This project reflects a long-standing personal connection to the genre and a practical goal of building something focused and complete. Its gameplay is familiar, but its priorities are different, using nontraditional tools and scenarios to reframe how players engage with each other and the game space.

Antillery's goal is not to solve a specific problem. It exists to preserve a format that still has value and to present it in a form that feels current without losing what made it enjoyable in the first place.

6.2 Market Need and Genre Stagnation

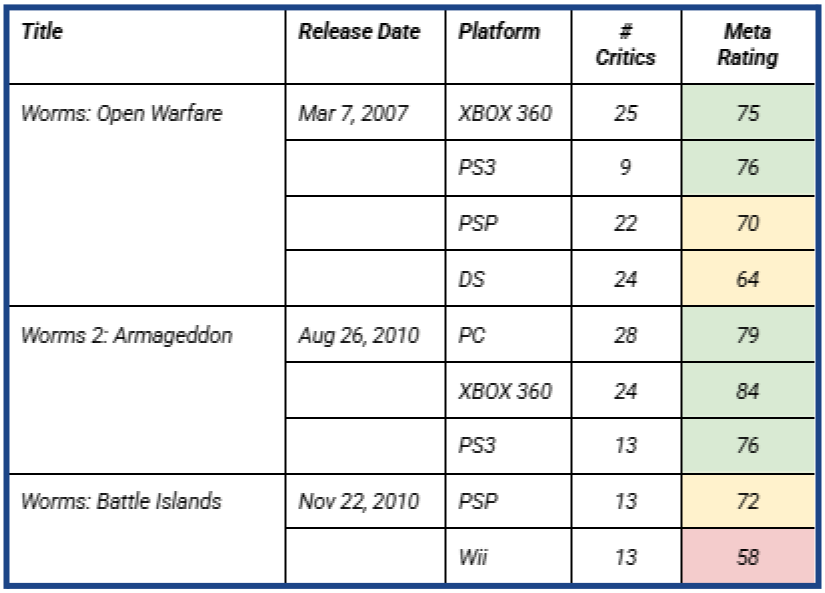
The artillery sub-genre holds a small but steady place within the turn-based strategy genre. While it continues to appeal to longtime players, its mechanical evolution has slowed in recent years. This slower pace allows projects like Antillery to revisit the format with a different perspective and renewed focus.

The Worms series, often seen as the face of the genre, has seen consistent releases since 1995. However, newer entries have received mixed responses depending on platform and direction. Table 6.2.1 highlights titles such as Worms: Open Warfare (2007) and Worms: Battle Islands (2010), which earned moderate Metacritic scores across multiple platforms. Worms Rumble (2020) introduced real-time mechanics instead of the traditional turn-based format. It received mixed reviews, including a score of 57 on Nintendo Switch. Although successful in some respects, the game stepped outside the genre rather than continuing it.

That shift does not mean artillery games have lost their place. It shows there is still room to build within the original framework, particularly for projects that take a different approach. Antillery builds on the core strengths of artillery gameplay, including local play, strategic timing, and terrain-based dynamics, while shifting focus toward cooperative tension and thematic contrast. It does not attempt to replace existing titles but offers a distinct alternative within a familiar structure.

6.2.1 A Selection of Metacritic Scores

Table 6.2.1 lists Metacritic critic scores for select Worms titles released between 2007 and 2020 to support the discussion of recent reception trends. The scores reflect platform-specific reviews and demonstrate various responses across entries and formats. A complete version of this table, including all Worms titles with available critic scores, appears in the Appendices.



Title	*Release Date*	*Platform*	*# Critics*	*Meta Rating*
Worms: Open Warfare	*Mar 7, 2007*	*XBOX 360*	*25*	*75*
		PS3	*9*	*76*
		PSP	*22*	*70*
		DS	*24*	*64*
Worms 2: Armageddon	*Aug 26, 2010*	*PC*	*28*	*79*
		XBOX 360	*24*	*84*
		PS3	*13*	*76*
Worms: Battle Islands	*Nov 22, 2010*	*PSP*	*13*	*72*
		Wii	*13*	*58*

Table 6.2.1. Selected Worms critic scores (Metacritic).

7 Who Is Antillery For?

Antillery invites players who enjoy strategy layered with unpredictability. It draws from the artillery genre's long history but repurposes familiar systems to explore less familiar outcomes. While some tools appear restorative or nonviolent, their true power depends on how the player uses them. Destruction might come from a missile or a well-placed sunflower. This blend of intent and consequence makes the game accessible to newcomers while rewarding mastery, mischief, and improvisation.

7.1 Longtime Artillery Fans

Players who grew up with Scorched Earth or Worms shaped the artillery genre through precision, chaos, and clever terrain use. They learned to expect indirect combat, physics-driven tools, and movement that doubled as personality. That history still defines how the genre feels today.

Antillery keeps those features in place: deformable maps, angled shots, limited mobility, and momentum-based action. However, it adds something unexpected. Some tools restore terrain or create cover, yet they can also trap enemies or limit their next move. Even a flower can cut off a retreat.

Antillery shifts meaning without changing the form. The shape of the game stays familiar, but its strategic depth expands.

7.2 Newcomers to the Genre

Not every player has experience playing artillery games. The skill-based barrier of entry might be challenging for some, with timed shots, weird angles, and no second chances. Antillery lowers the barrier without sacrificing depth.

It teaches through play. Turns move fast, aiming feels intuitive, and outcomes are visible. Peace tools soften early mistakes by keeping ants alive longer. These tools act like training wheels but still carry risk. A misplaced flower can block an escape or flip the match.

By offering tools that teach without preaching, Antillery brings new players in without flattening the experience. The genre stays sharp, but now more players can reach its depth.

7.3 Casual Gamers

Some players want quick games, unpredictable outcomes, and moments worth laughing about. Antillery welcomes that energy without making the experience disposable.

Rounds move fast, but every choice still carries weight. Peace tools shift the flow in ways that explosions might not, stalling a turn, blocking a shot, or setting up accidental outcomes. Players who experiment can win by planning or turning chaos into a strategy.

Antillery gives casual players room to play without pressure. The game stays fun whether someone is aiming for mastery or just trying to mess with their friends.

7.4 Competitive Players

Players who seek mastery look beyond explosions. Antillery offers layered complexity where every decision, including restraint, carries weight.

Winning depends on more than aim. Terrain can be shaped or repaired. Players can outsmart their opponent into wasting a turn. Some tools pressure through presence alone, setting traps, or forcing a mistake without firing a shot. Competitive players learn to control space, deny momentum, and plan turns ahead.

Antillery rewards those who think past raw damage. The strongest matches often hinge on who considered both sides of combat, not who fired first.

7.5 Conclusion: Strategy Without a Fixed Form

Antillery is not one game played one way. It adjusts to the player, whether they chase chaos, control, mischief, or mastery.

The rules stay simple. Shots behave consistently, and movement follows a predictable arc, but outcomes shift based on what each player hopes to build, block, or break. Even peace tools bend under pressure.

The result is a strategy game without deterministic outcomes. Destruction and restoration share inputs. Meaning comes from how players use them.

8 Vision Statement

8.1 Subversive Nostalgia

8.1.1 Vision

Reinvigorating turn-based artillery through eco-satire and strategic irony.

8.1.2 Statement

To revive the artillery genre by pairing its legacy of destructive chaos with tools of peace, allowing players to reshape strategy through irony, absurdity, and environmental subversion.

8.2 Weaponizing Whimsy

8.2.1 Vision

Redefining combat with wildflowers, restraint, and passive resistance.

8.2.2 Statement

To challenge what counts as a weapon by introducing mechanics rooted in peace. Antillery lets players wage war with misdirection, humor, and antagonistic passivity, turning gentle gestures into tactical disruption.

8.3 Art of War

8.3.1 Vision

Victory isn't always red; sometimes, it's green, quiet, or absurd.

8.3.2 Statement

To offer a tactical experience that values restoration as much as destruction. Antillery expands classical artillery mechanics into a creative sandbox where peace can punish, and strategy grows wild.

9 Elevator Pitches

9.1 Fans of Artillery Games

Antillery puts players in charge of a team of cheeky ants waging strategic mayhem across a destructible ant farm. Inspired by the satirical spirit of Worms, Antillery blends classic turn-based artillery combat with peace-driven chaos and imaginative tools like the Tree-Zooka and Luster Bombs.

9.2 Newcomers to Artillery Games

Antillery is not your grandfather's artillery game. It keeps the explosions but rewires the battlefield with peace-driven chaos and strategic mischief. No experience with artillery games is needed–just a sense of humor and a taste for absurdity.

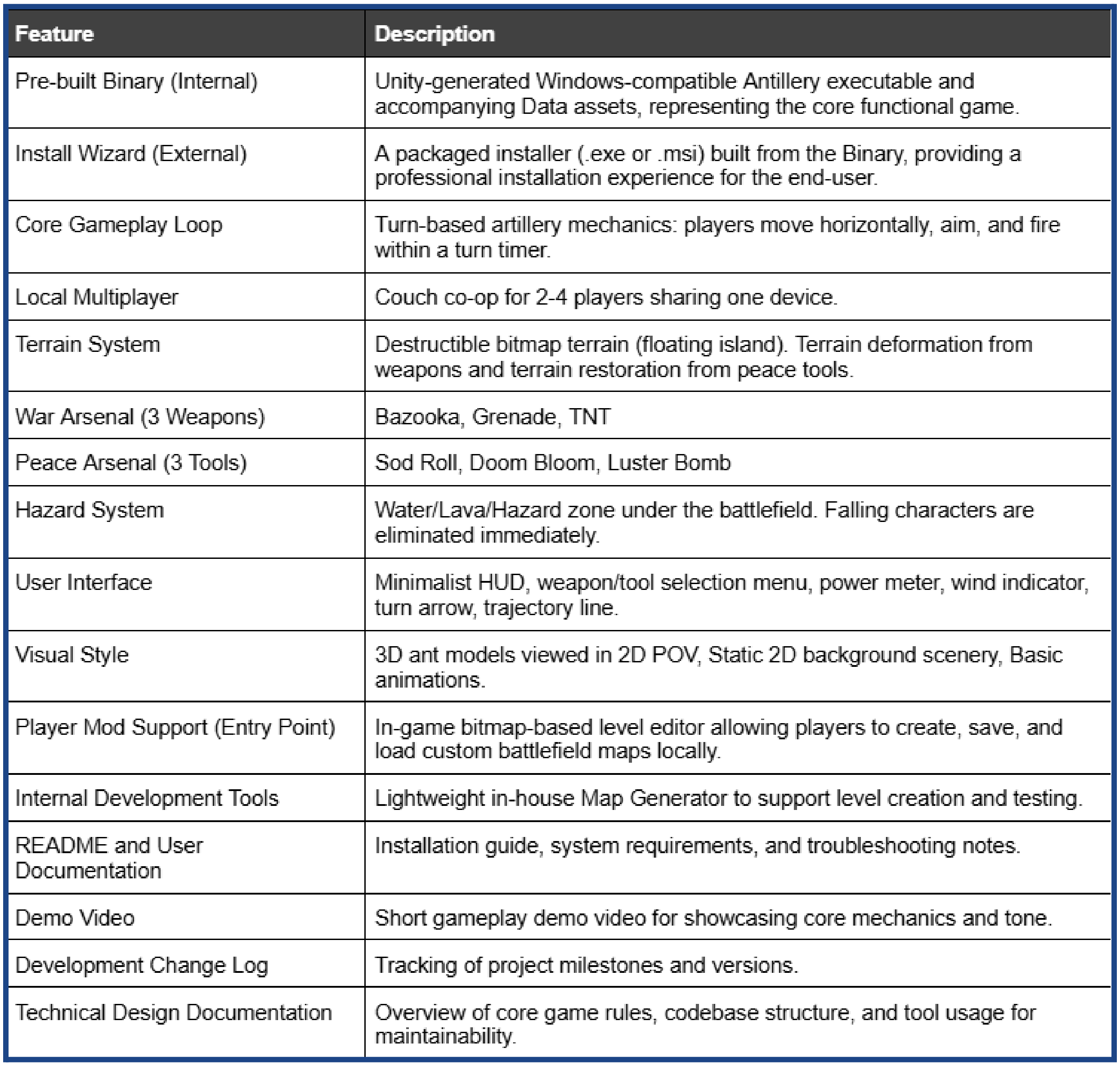
9.3 Potential *Antillery* Investors

The artillery genre hasn't seen much real progress in years. Worms, the biggest name in the space, has mostly leaned on remasters and re-releases. Antillery offers a new angle, built with scalability, creativity, and modding at its core. There's still a community that wants more–and Antillery aims to deliver it.

10 Scope

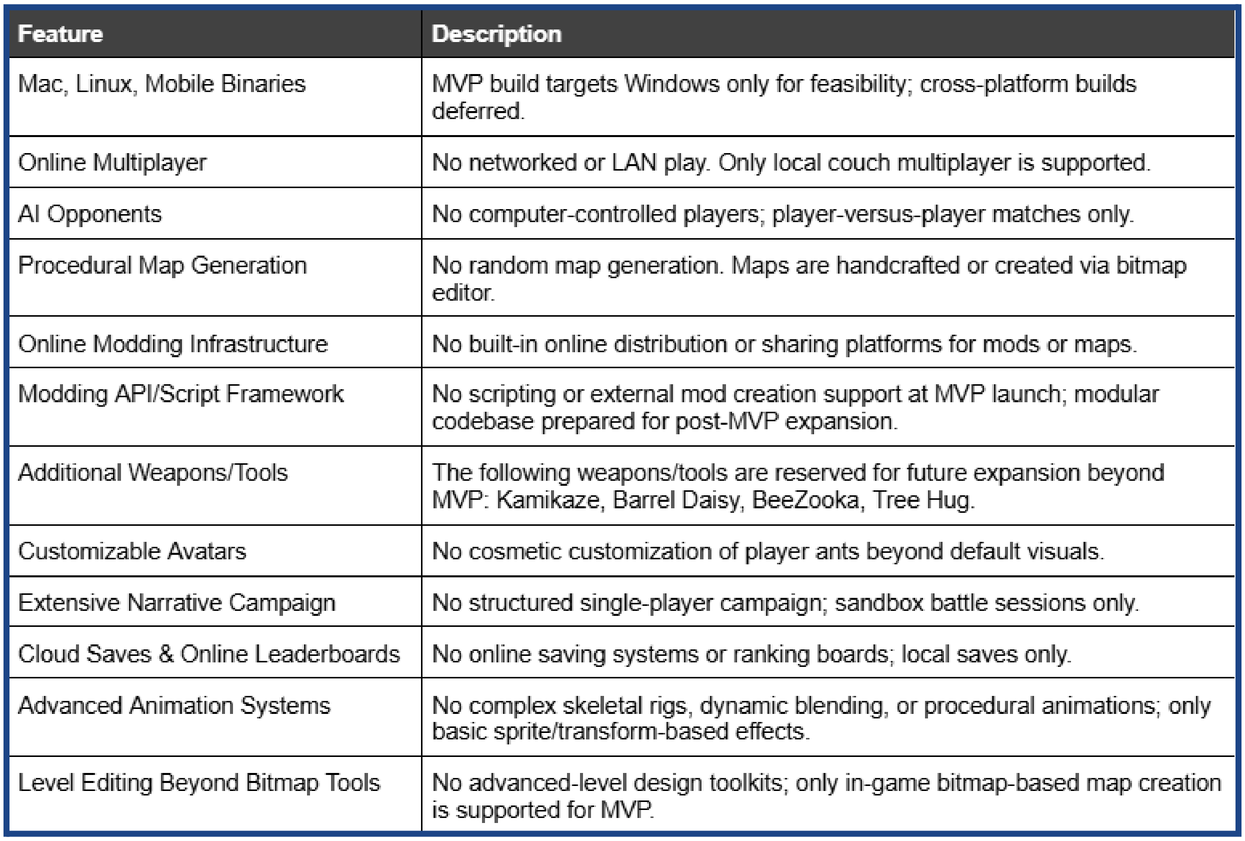
This section defines the boundaries of Antillery's Minimum Viable Product (MVP). It details which core features the project will develop and deliver and which it will explicitly exclude. The selected MVP scope is intentionally conservative to ensure feasibility for a solo developer within the academic timeframe while establishing a strong, expandable foundation for future enhancement.

10.1 In Scope



Feature	Description
Pre-built Binary (Internal)	Unity-generated Windows-compatible Antillery executable and accompanying Data assets, representing the core functional game.
Install Wizard (External)	A packaged installer (.exe or .msi) built from the Binary, providing a professional installation experience for the end-user.
Core Gameplay Loop	Turn-based artillery mechanics: players move horizontally, aim, and fire within a turn timer.
Local Multiplayer	Couch co-op for 2-4 players sharing one device.
Terrain System	Destructible bitmap terrain (floating island). Terrain deformation from weapons and terrain restoration from peace tools.
War Arsenal (3 Weapons)	Bazooka, Grenade, TNT
Peace Arsenal (3 Tools)	Sod Roll, Doom Bloom, Luster Bomb
Hazard System	Water/Lava/Hazard zone under the battlefield. Falling characters are eliminated immediately.
User Interface	Minimalist HUD, weapon/tool selection menu, power meter, wind indicator, turn arrow, trajectory line.
Visual Style	3D ant models viewed in 2D POV, Static 2D background scenery, Basic animations.
Player Mod Support (Entry Point)	In-game bitmap-based level editor allowing players to create, save, and load custom battlefield maps locally.
Internal Development Tools	Lightweight in-house Map Generator to support level creation and testing.
README and User Documentation	Installation guide, system requirements, and troubleshooting notes.
Demo Video	Short gameplay demo video for showcasing core mechanics and tone.
Development Change Log	Tracking of project milestones and versions.
Technical Design Documentation	Overview of core game rules, codebase structure, and tool usage for maintainability.

10.2 Out of Scope



Feature	Description
Mac, Linux, Mobile Binaries	MVP build targets Windows only for feasibility; cross-platform builds deferred.
Online Multiplayer	No networked or LAN play. Only local couch multiplayer is supported.
AI Opponents	No computer-controlled players; player-versus-player matches only.
Procedural Map Generation	No random map generation. Maps are handcrafted or created via bitmap editor.
Online Modding Infrastructure	No built-in online distribution or sharing platforms for mods or maps.
Modding API/Script Framework	No scripting or external mod creation support at MVP launch; modular codebase prepared for post-MVP expansion.
Additional Weapons/Tools	The following weapons/tools are reserved for future expansion beyond MVP: Kamikaze, Barrel Daisy, BeeZooka, Tree Hug.
Customizable Avatars	No cosmetic customization of player ants beyond default visuals.
Extensive Narrative Campaign	No structured single-player campaign; sandbox battle sessions only.
Cloud Saves & Online Leaderboards	No online saving systems or ranking boards; local saves only.
Advanced Animation Systems	No complex skeletal rigs, dynamic blending, or procedural animations; only basic sprite/transform-based effects.
Level Editing Beyond Bitmap Tools	No advanced-level design toolkits; only in-game bitmap-based map creation is supported for MVP.

The defined scope for Antillery prioritizes the delivery of a polished, playable Minimum Viable Product that balances core gameplay innovation with solo development feasibility. This strategic containment measure ensures the project remains feasible within the academic timeframe while staying true to the goal of evolving the artillery genre.

11 Core Gameplay Mechanics

This section outlines the mechanical systems that define Antillery's gameplay experience. It includes the destructive and constructive arsenals available to players, key environmental interactions, and the structure of player turns.

11.1 War Tools: Destructive Arsenal

11.1.1 Bazooka

The **Bazooka** functions as a high-velocity projectile weapon. Players determine their trajectory and distance by adjusting the angle and charge time. Upon impact, it causes moderate terrain deformation and direct damage to any players within its blast radius.

Figure 11.1.1. Bazooka Ant Concept

11.1.2 Grenade

The **Grenade** is a timed explosive that incorporates bounce physics. Players control throw strength similarly to the **Bazooka**. Once deployed, the **Grenade** detonates after a countdown regardless of its final position. This tool is especially effective for reaching confined areas and executing indirect attacks.

11.1.3 TNT

TNT is a fixed-placement explosive that activates after a short delay. It requires proximity to the target to be effective. When triggered, it delivers significant damage to terrain and nearby players. This tool is most useful for close-range plays or removing key terrain features.

11.2 Peace Tools: Constructive Arsenal

11.2.1 Sod Roll

The Sod Roll launches a roll of pre-grown sod that unfurls across the terrain, restoring ground as it moves. Speed and range depend on charge time. Deploying a partial roll can create barriers. Colliding with an ant causes light damage and ends the roll.

11.2.2 Doom Bloom

The ant buries itself, sacrificing a turn to sprout a destructible sunflower that forms a bridge or canopy. This tool is a peaceful tool that provides shelter, bridges gaps, and can block opponents.

11.2.3 Luster Bomb

The Luster Bomb releases a cloud of butterflies that slows projectiles. Positioned correctly, it deflects or absorbs enemy fire, protecting ants below and altering combat flow.

11.3 Battlefield Systems

Antillery’s terrain is fully destructible. Explosive tools carve or remove terrain pixels, while peace tools restore or add new segments. These modifications affect movement, visibility, and long-term strategy as terrain changes persist between rounds and accumulate over time.

11.4 Turn Structure & Player Actions

Players have a limited move period in each turn and can use a single weapon or tool. Once a tool is activated or placed, the turn ends. Cooldowns and limited ammunition promote strategic pacing and resource management.

12 Feasibility

Ensuring that Antillery is achievable within the given time frame, resource constraints, and technical expectations requires a careful assessment of feasibility. This section demonstrates that the project's complexity is well understood, the developer's skill set sufficiently meets the challenge, the scope is realistic, and the production timeline fits within the 16-week semester.

12.1 Project Complexity: Overcoming Game Architecture Pitfalls

One challenge that needs meticulous care in game software development is managing the interdependence of systems. Poor planning can result in tight coupling among the many layers, leading to scalability issues, debugging complications, and compromise testing. These issues can quickly derail an otherwise promising project.

Antillery avoids these pitfalls through intentional architectural choices. The design emphasizes modularity and separation of concerns by leveraging Unity's prefab system, serialized ScriptableObjects, and behavioral patterns such as the State Pattern, as described in the Gang of Four's Design Patterns (Gamma 305). These strategies minimize coupling, reduce cross-system fragility, and support maintainability.

This modular, layered approach reflects standard industry practices. For reference, Jason Gregory's Game Engine Architecture outlines how large-scale engines segment functionality into manageable layers to reduce dependency and improve debugging clarity (Gregory 33). While Antillery operates on a much smaller scale, the core architectural principles remain applicable.

No software design can account for every scenario, but Antillery applies proven principles from industry practice and personal project experience. The system's structure prioritizes flexibility, scalability, and resilience to common development pitfalls.

12.2 Developer Skill Set: Matching Experience to Project Scope

A project of this complexity requires a practical foundation in software architecture, programming, and project management, built through years of academic coursework and independent game development projects. The resulting skillset satisfies the engineering demands of this capstone.

Prior experience centers on Unity, C++, and C# game programming. The Unity-based 3D rail shooter prototype (Gal Game) demonstrates practical project management and Agile development practices through accompanying development logging. The project Server

Surfer(Neon Cloud) also showcases a game architecture built on design patterns and Unity's prefab system. A custom arcade engine in C++ using SFML showcases modular architecture, a fully developed state-driven game loop, and persistent high-score serialization (Lollis, "Legacy Portfolio Site").

These projects reflect fluency in modular game architecture and reinforce skills developed through coursework in game design, algorithm engineering, and software engineering. Coursework in The Principles of Computer Graphics further supplemented this game development-focused skill set with experience in 3D asset production. As seen below, the final project demonstrated modeling in 3DS Max, rendering with Chaos Corona, and advanced lighting concepts like reflection and caustics (Lollis, "Legacy Portfolio Site").

This diverse background reflects the complexity, depth, and self-direction required for an ambitious solo capstone project like Antillery.

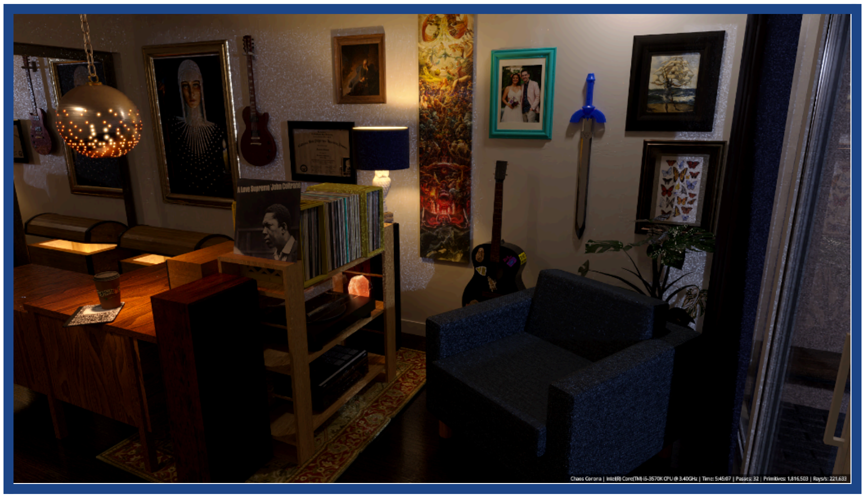


Figure 12.2. Final render for CPSC 484: Principles of Computer Graphics (Lollis).

12.3 Resource Constraints: Balancing Team Size and Scope

The greatest threat to the feasibility of Antillery is its limited development team. Game development typically involves a tradeoff between personnel count and time-to-completion. Developing a video game requires the coordination of many specialized roles–designers, programmers, artists, composers, and QA testers. Even small projects can become unmanageable when the scope is not carefully constrained.

This project maintains a well-defined minimum viable product (MVP) that excludes nonessential features such as AI and network functionality. Consultation with faculty confirms that the project remains within the appropriate complexity for a capstone project. Prior completion of similarly scoped projects further supports confidence in the proposed timeline. A single developer is performing every role required to complete Antillery. By contrast, the original Worms title involved at least 25 individuals in its completion (Team17 9).

In summary, the project's scope is not arbitrarily limited but proportionate to available resources and time, supporting its feasibility. By intentionally narrowing the scope, Antillery mitigates risk stemming from team size.

12.4 Timeline Feasibility: Delivering a Game in 16 Weeks

The primary objective is to deliver a beta version within the 16-week semester. Problem recognition and solution planning usually take at least the first few weeks of a project; the proposal handles much of that upfront.

The Activities section outlines a sprint-based development plan that allows approximately 12-15 hours per week for development; the total hours needed amounts to 190-240, which is on par with the workload of past solo Unity projects. This structure enables efficient, iterative development from the first week by addressing early planning, architecture, and risk scoping upfront.

For context, the original Worms game followed a multi-year development cycle. Andy Davidson began prototyping Total Wormage in 1993 and released the final version with Team17 in late 1995. Davidson developed Worms without modern engines or asset pipelines and coded them in BASIC on an Amiga (Nuttall 26). Antillery has a much narrower timeframe but benefits from Unity's robust tooling, asset integration, and build pipeline, significantly reducing development overhead.

While the workload is substantial, careful planning, a constrained scope, and prior development experience make a 16-week development cycle realistic and strongly supported by schedule design and project precedent.

13 Objectives

The objectives of this capstone project outline the concrete, measurable, and realistic outcomes expected from *Antillery's* development. These objectives not only guide the semester's development milestones but also reflect personal and professional goals within the scope of the academic timeline.

13.1 Deliver a Complete Version 1.0

The primary objective of *Antillery* is to release a fully playable, in-scope Version 1.0 by the final project deadline.

The Minimum Viable Product (MVP) will include all core features listed in the Scope section: functional local multiplayer, balanced weapons, an intuitive user interface, destructible and repairable terrain, and projectiles affected by environmental factors. These systems establish the minimum standards to define a "complete" and playable experience.

Successfully achieving this objective guarantees the academic validity of the project and provides a clear "definition of done" for tracking progress and managing scale effectively.

13.2 Establish the Foundation for a Scalable Passion Project

In addition to completing the MVP, *Antillery* aims to serve as the foundation for a long-term passion project built on carefully designed, scalable architecture and modular design principles.

The project will support future experimentation, technical growth, and potential collaboration by leveraging proven software design patterns, such as the State pattern, and writing flexible, reusable code. Though outside the current semester's scope, future iterations may include networked play, platform abstraction, or extensible tools for player modding. Laying a strong foundation now will reduce scaling complexity later.

This objective reframes *Antillery* from a one-off academic obligation into a sustainable sandbox for ongoing creative expression and technical experimentation.

13.3 Strengthen Game Development Portfolio

A key goal of this project is to create a portfolio-ready game that exemplifies bachelor's-worthy CS knowledge and specialized skills in industry-grade game development practices.

The project reflects real-world workflows by developing in Unity, a widely used, industry-standard engine. *Antillery* will demonstrate skills ranging from systems architecture and

C# scripting to UI/UX design, player experience crafting, asset creation, and audio composition. This project will be a well-rounded technical piece and a multi-disciplinary indie developer feat.

When complete, *Antillery* will be a portfolio strengthener and demonstrate readiness for internships, indie studio collaboration, or employment in the game industry.

13.4 Innovate in a Genre of Personal Interest

Team17 once led the way in 2D artillery games, but their shift toward remasters and collaborations has left a noticeable gap in the genre. Fans of the chaotic and irreverent Worms series still feel that absence. The *Worms World Party* Remastered Web Banner (Figure 13.4) below illustrates this shift. (Team17)

Antillery aims to pick up where that legacy left off. It draws heavily from *Worms World Party*, a title with significant personal nostalgia. The goal is not to replace *Worms* but to carry its spirit forward, keeping the humor and satire alive while updating the experience with new mechanics and a sharper modern cultural edge.

This project is a nod to what made *Worms* great and a push to evolve the formula for a new generation. It is about honoring what came before while creating something that feels modern.

Figure 13.4. *Worms World Party Remastered* Web Banner. Team17.

14 Targeted Problem

Antillery responds to specific gaps in the artillery game genre and modern gaming culture. The problems identified here directly inform the objectives and project scope, each defined previously.

14.1 Genre Stagnation

The 2D artillery genre has stagnated. Worms redefined the space decades ago, but recent titles have repeated the same mechanics with minimal variation. Without new systems or a shift in tone, the genre risks fading into nostalgia.

While Team17's original Worms (1995) redefined the space with playful destruction and expressive mechanics (Barton), its recent output leaned heavily on remasters. This approach locks the genre into its past instead of pushing it forward. Antillery moves in a different direction by bringing new themes and giving players tools to shape how the game plays. A built-in map editor ships with the game, letting players design terrain from the start, allowing unique battles to emerge. Full modding support follows later in the development roadmap. Scacchi notes that developer-laid foundational frameworks and community-created content invigorate genre vitality (Scacchi). Giving players agency to define what artillery means is an act of evolution.

Preserving a genre requires more than homage. It requires permeability and room for new voices, tools, and subversions. Antillery treats stagnation not as a heritage to be protected but as an opportunity to expand the genre's meaning.

14.2 Online-Only Saturation

The dominance of online-only games in the modern gaming industry has led to various issues, ranging from the loss of local multiplayer culture to exploitative monetization systems. *Antillery* responds by embracing a design philosophy of real-world interaction, accessibility, and transparency.

14.2.1 The Death of Couch Multiplayer

Couch multiplayer has taken a backseat to matchmaking queues and online anonymity. Nevertheless, many players' desire for in-person, glee-filled chaos remains strong.

The decay of local multiplayer has been a grave cultural loss. It is more than just a gimmick or revenue-generating feature; it is a social bonding ritual important to gaming culture. Many gamers today cling dearly to the memories tied to these experiences. The nostalgia of sleepovers spent around friends in shared-screen mayhem is what inspires *Antillery*'s purpose.

Local multiplayer is not dead; the AAA game Industry has instead fine-tuned the machine generating the most revenue. *Antillery* aims to carry the torch for the cultural importance of couch multiplayer in gaming.

14.2.2 Always-Connected Frustration

Constant online requirements can gatekeep players who lack access to stable internet. *Antillery* offers an alternative that respects accessibility and believes in ownership of the software a player pays for.

Not all players have consistent internet access, especially in low-income, rural, or underserved areas. Moreover, even for those who do, the requirement to always be online creates unnecessary hassle; servers go down, login authentication is tedious, and single-player experiences become hostage to factors outside the player's control. *Antillery* sidesteps these frustrations by allowing players to play completely offline with local multiplayer as a first-class feature, respecting players' ownership of the software they paid to play.

A game should not require a server handshake to let friends sit and play. *Antillery* removes this gatekeeping and returns the focus to players, not network tedium.

14.3 Modding as a Livelihood, Not a Liability

Modding can provide a rich sandbox for creative expression and community building. It should be embraced and fostered as a vital part of the lifespan and legacy of a game.

Some studios treat mods as liabilities, threats to intellectual property, or lost monetization opportunities. Others, like Bethesda, have cultivated modding into full-blown subcultures. Skyrim (2011) did not survive exclusively through official patches; it thrived because modders transformed a static game into thousands of distinct, playable experiences. Nearly 100,000 mods later, it remains relevant in 2025. Mods fill gaps developers never addressed. They allow players to reshape the game in their image, iterate on design, and stretch core systems to their limits. Whether for accessibility, absurdity, or artistry, modding is a tool for player-driven authorship and cultural longevity. As Scacchi notes, it is also a common first step into professional game development, equipping creators with technical skills and real-world experience (Scacchi).

Antillery supports modding from the start, not as an afterthought. Players are not passive users but collaborators with unique intent and expression. The game is a platform, not a black box. Antillery invites players to breathe their unique life into it.

15 Activities

The development of *Antillery* will follow a 16-week Agile methodology, with each week representing a spring. Each sprint will introduce or improve upon a system, mechanic, interface component, or improvement task. These weekly sprints will set a pace to ensure project completion and organization.

15.1 Week 1: Project Initialization

- Create a Unity project with a versioned folder hierarchy
- Setup GitHub repo and enable Git LFS for Unity assets
- Initialize the Kanban board with sprint columns and core tasks
- Create design document scaffolds: MVP checklist, coding conventions, asset pipeline

15.2 Week 2: Core Systems Architecture

- Implement core game-state and turn-state machines
- Integrate event-driven architecture for turn handling
- Define project-wide enums and ScriptableObject data models (weapon/tool definitions)
- Begin technical documentation for the codebase

15.3 Week 3: Movement & Input Layer

- Implement player input and per-turn movement limits
- Set up Unity Input System with action maps
- Basic UI feedback: directional arrows and movement readiness indicator

15.4 Week 4: Projectile System & Aiming

- Implement aiming mechanics and trajectory controls (angle, power)
- Basic projectile instantiations and travel arc
- Simulate collisions and test destructible terrain hooks

15.5 Week 5: Destructible Terrain System

(subject to revision based on integration considerations with Unity's collision system specifics.)

- Implement bitmap-based terrain using a black/white mask texture
- Apply real-time alpha-based pixel removal upon impact
- Rebuild mesh collider after terrain mutation
- Integrate terrain change logic with projectile and hazard systems.

15.6 Week 6: Environmental Forces & Physics

- Add wind system affecting projectile trajectories
- Finalize physics parameters: gravity, drag, impact force
- Display wind vectors using onscreen HUD indicators

15.7 Week 7: War Weapon Implementation

- Implement Bazooka, Grenade, and TNT mechanics
- Add distinct audio/visual feedback and impact animations
- Tune explosion radius and test terrain deformation integrity

15.8 Week 8: Weapon System (Peace-based)

- Implement Sod Roll, Doom Bloom, and Luster Bomb mechanics
- Ensure Sod Roll mechanics work properly in edge cases
- Ensure visual and audio effects distinguish tone from war tools

15.9 Week 9: User Interface Systems

- Build weapon/tool selection HUD and contextual menus
- Implement the power meter, turn timer, and wind indicators
- Validate real-time feedback for all core turn actions

15.10 Week 10: Map Editor and Level System

- Build in-game bitmap-based map editor using brush and stamp tools
- Enable saving and loading of custom PNGs from local storage
- Validate file I/O and bitmap

15.11 Week 11: Art Integration and Animation Systems

- Import 3D ant models and set orthographic 2D camera rig
- Configure idle, walk, aim, and fire animations
- Integrate state transitions using Unity Animator and gameplay triggers

15.12 Week 12: Multiplayer Setup (Local)

- Implement a round-robin player-switching system
- Track player state across turns and reset between rounds
- Test gameplay for 2-4 players on a shared device

15.13 Week 13: Internal Dev Tools and Debug Features

- Bild internal Map Generator for rapid test scenario creation
- Create debugging overlays (e.g., collision boxes, terrain debug mask)
- Add shortcut keys for level reset, weapon cycling, and log output

15.14 Week 14: Optimization & Profiling

- Profile memory use and garbage collection during gameplay
- Optimize terrain mutation, physics checks, and render performance
- Refactor redundant scripts and batch costly operations

15.15 Week 15: Playtesting & Polish

- Conduct internal playtests and log balance/UX notes
- Address gameplay issues, pacing, terrain abuse, and unclear interactions
- Final pass on the audio mix, visual clarity, and turn flow transitions

15.16 Week 16: Release Preparation

- Export the final Windows build using Unity's build pipeline
- Package installer and create a release branch with MVP tag
- Assemble README, documentation, and a gameplay demo video

16 Development Environment

16.1 Computers

The development will primarily take place on a desktop PC, but using a laptop for light tasks is likely. Laptop tasks include drafting documentation and developing internal tools that do not require Unity.

16.1.1 Desktop Specifications

- **CPU:** Intel Core i5-14400F (1.8GHz)
- **RAM:** 32GB DDR5-6000
- **GPU:** NVIDIA GeForce RTX 4060 Ti (8GB)
- **Storage**: 1TB SSD
- **OS:** Windows 11 Professional

16.1.2 Laptop Specifications

- **OS:** Ubuntu or Windows 11 (depending on CEDA availability)

16.2 Operating System

Windows 11 Professional

Windows 11 Professional will be the primary operating system for development, currently installed on the main development machine. This environment will support most project activities, including development in Unity Editor, build pipeline management, and final executable packaging for the Windows MVP release.

Ubuntu Linux

Ubuntu may fill a secondary role, depending on the equipment availability at CEDA during the development phase. It will primarily be for lightweight programming tasks and documentation drafting, if available.

16.3 Software & Tools

16.3.1 Integrated Development Environments (IDEs)

- **Unity Editor (Latest LTS Version)** – Main platform for game development.
- **Visual Studio** – Used for Unity scripting and debugging via Unity's integration.

- **Visual Studio Code** – Used for Markdown documentation and lightweight coding tasks.
- **QT Creator** – Used for building internal development tools.

16.3.2 Version Control and Project Management

- **GitHub** – Version control and backup, using Git LFS for large Unity assets.
- **Branching Strategy:**
    - **main** – stable release
    - **dev** – ongoing development
    - **feature** – feature or experimental branches
- **GitHub Projects** – Used to manage tasks, bugs, and milestones.

16.3.3 Creative and Support Tools

- **Blender** – For 3D modeling and animation, where applicable.
- **Adobe Suite**
    - **Photoshop** – For 2D asset creation and texture editing
    - **Illustrator** – For UI elements and vector graphics.
    - **Premiere** – For demo reel editing and video production.
    - **Express** – For the rapid design of promotional visuals.
    - **Creative Cloud** – For asset synchronization, versioning, and backup.
- **Google Suite**
    - **Docs** – For general writing tasks.

16.4 Languages, Libraries, & Frameworks

- **C#** – For all Unity-related development, and some internal tools.
- **C++ / Python / JavaScript / QML** – Used in internal development tools.
- **Unity Engine** – Primary engine framework for the game.
- **Unity Packages** – TBD as needs arise (e.g., Input System, URP).
- **QT Framework** – For standalone tools that benefit from cross-platform design.

16.5 Backup and Security

- **GitHub with Git LFS** – Stores all Unity source code and large project files.
- **GitHub VCS** – Also used for internal tool and script backups.
- **Google Drive** – Stores any non-GitHub tracked documents.
- **Adobe Creative Cloud** – Manages backup and versioning of art, video, and other creative assets.

17 Products

17.1 Final Game Build

17.1.1 Executable

- A standalone Windows-compatible build of Antillery, packaged as an installer or portable executable (.exe) using Unity's build pipeline.
- The game will support 2+ player local multiplayer with all MVP features implemented.

17.2 Development Tools

17.2.1 Map Generator

- A simple internal-use tool for generating terrain layouts to expedite playtesting and level variation.
- Allows for quicker debugging, testing, and balance checks during development.
- May be released as part of the mod support objective.

17.3 Marketing and Presentation

17.3.1 Demo Video

- A short gameplay trailer or feature walkthrough video showing off the core mechanics and tone of the game.
- Intended for sharing with stakeholders.
- Valuable for portfolios.
- Includes brief clips of different weapon/tool uses, terrain dynamics, and UI.

17.4 Documentation

17.4.1 README

- Instructions on how to install and launch the game.
- Specifies minimum system requirements.
- Specifies recommended system requirements.
- Includes FAQ or troubleshooting notes.

17.4.1 Change Log

- A concise chronological list of updates and fixes made during development.
- Used to track progress across sprints and highlight version history.
- May include balance tweaks, feature additions, bug fixes, or known issues.

17.4.3 Design & Technical Documentation

- Internal-use documents explaining:
  - Game rules and mechanics
  - Codebase structure or notable architectural decisions.
  - Tool usage (e.g., Map Generator)
- This helps future developers, contributors, or instructors understand how the game was built and the intentions behind various decisions.

18 Risks & Mitigations

18.1 Incomplete Feature Implementation

Antillery's architecture combines complicated systems, including physics simulation, animation state management, game-flow state management, and dynamic environments. Their complexity creates significant architectural challenges. Since a single developer is building the project under a limited timeframe, there is a heightened risk of not completing all features on time.

When one system breaks, it can easily cause a chain reaction. For example, a bug in projectile physics might throw off collision detection, which could affect animations or even turn transitions. These cascading problems present significant risks in a solo project because identifying the root cause can consume precious time.

To keep the project on track, careful feature prioritization, clear short-term goals, and the flexibility to adjust plans quickly are essential. Without an active strategy to manage these risks, delivering a Minimum Viable Product (MVP) on time would become much more difficult.

18.1.1 Strategy 1: Feature Prioritization

Carefully prioritizing features is one of the best ways to prevent incomplete systems from delaying the project. The key to this strategy is assessing how closely each feature ties into the core architecture, how many other systems depend on it, and whether it is critical for completing the Minimum Viable Product (MVP) on time.

Essential systems such as the game-flow state machine, basic player movement, turn-based mechanics, and projectile physics must precede nonessential polish features like animation or particle effects. This strategy guarantees that the MVP works well and is complete, even if features change.

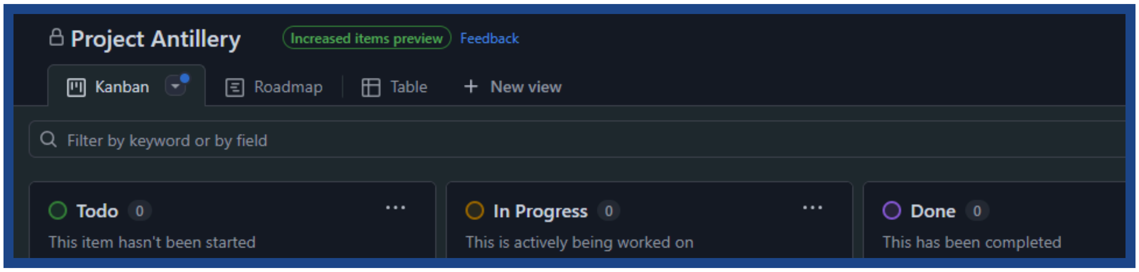
The project follows the idea: "Finish small tasks often." This strategy helps the codebase stay resilient to unexpected problems and increases the likelihood of delivering it on time.

18.1.2 Strategy 2: Utilize Agile Development

Using Agile development practices helps manage project risk by focusing on steady, realistic progress. Instead of following a rigid, step-by-step plan, Agile encourages breaking work into smaller pieces, setting short-term goals, and adjusting plans based on what gets done.

Tools like GitHub Projects Kanban board, roadmap, and table features (see Figures 18.1.2.1-3 below) help track tasks and identify potential problems early. Regularly scheduled Scrum meetings and progress assessments help maintain accountability and development direction, preventing minor issues from becoming larger setbacks and making it easier to adjust features as priorities change.

This strategy lowers the chances of unanticipated last-minute setbacks and builds momentum through small, completed milestones. Staying "Agile" helps ensure the MVP will be finished on time, even if some adjustments are needed. These potential adjustments are made possible through Agile development.



Todo 0
This item hasn't been started

In Progress 0
This is actively being worked on

Done 0
This has been completed

Figure 18.1.2.1. GitHub Projects Kanban board feature.

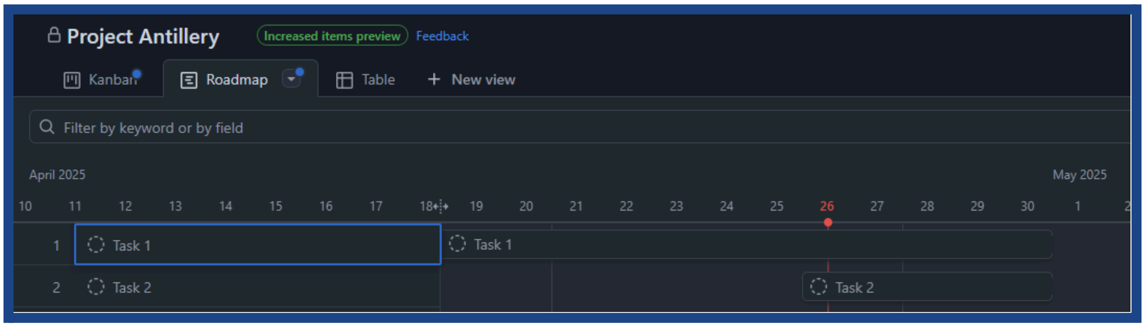
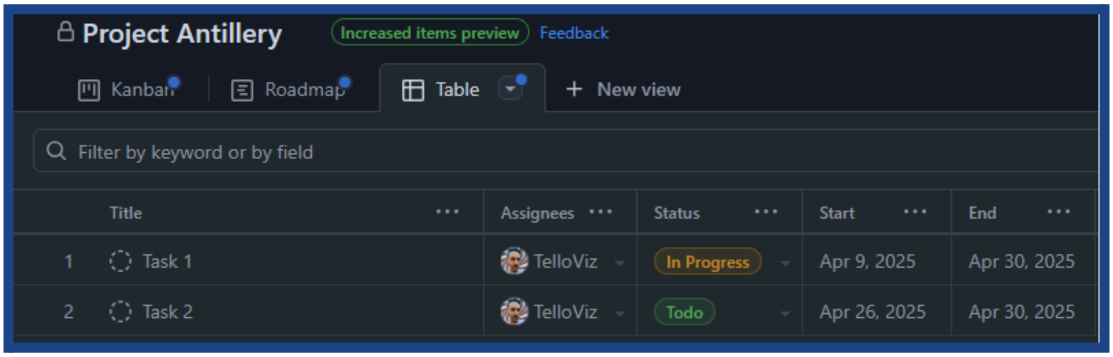


Figure 18.1.2.2. GitHub Project's roadmap feature.



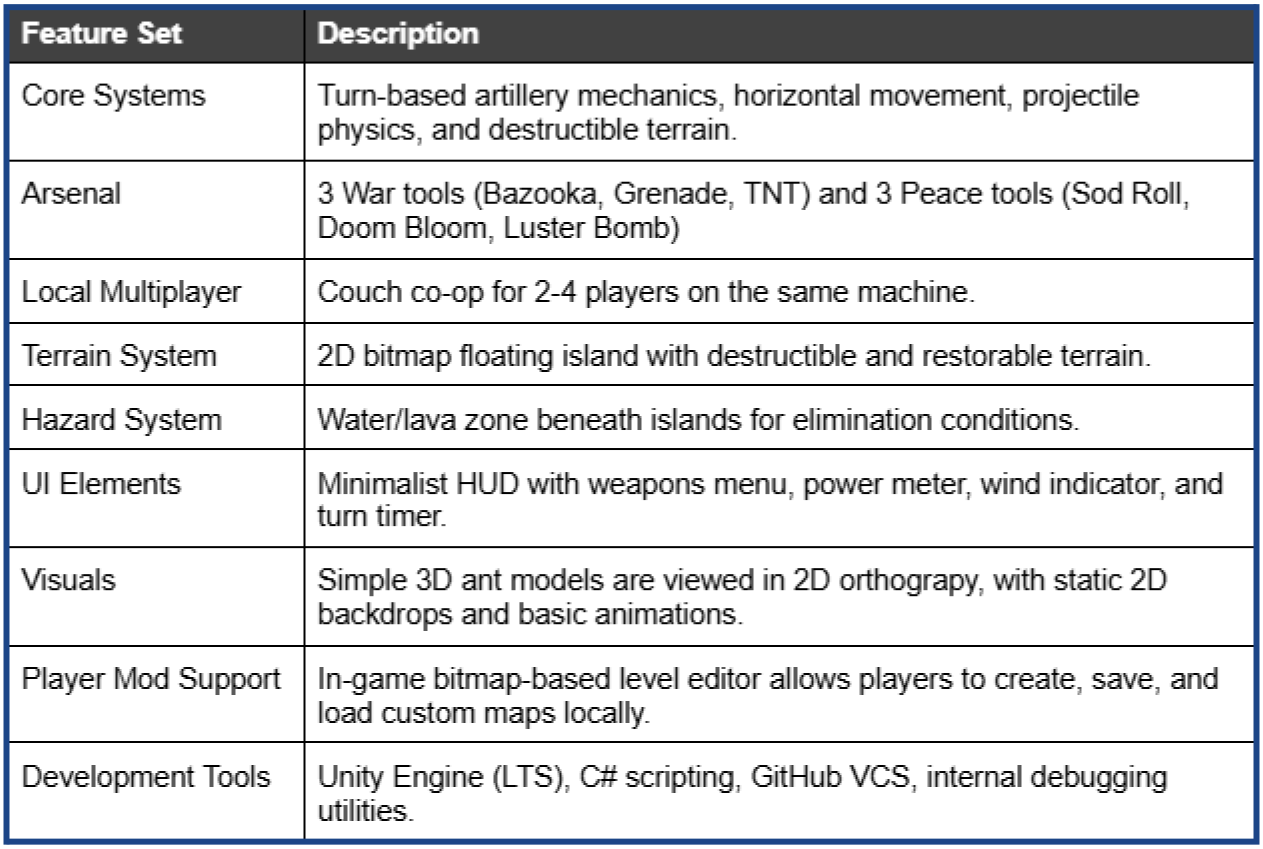
	Title	Assignees	Status	Start	End
1	Task 1	TelloViz	In Progress	Apr 9, 2025	Apr 30, 2025
2	Task 2	TelloViz	Todo	Apr 26, 2025	Apr 30, 2025

Figure 18.1.2.3. GitHub Project's table feature.

19 Release Plan

19.1 Antillery Beta (Capstone MVP Release)

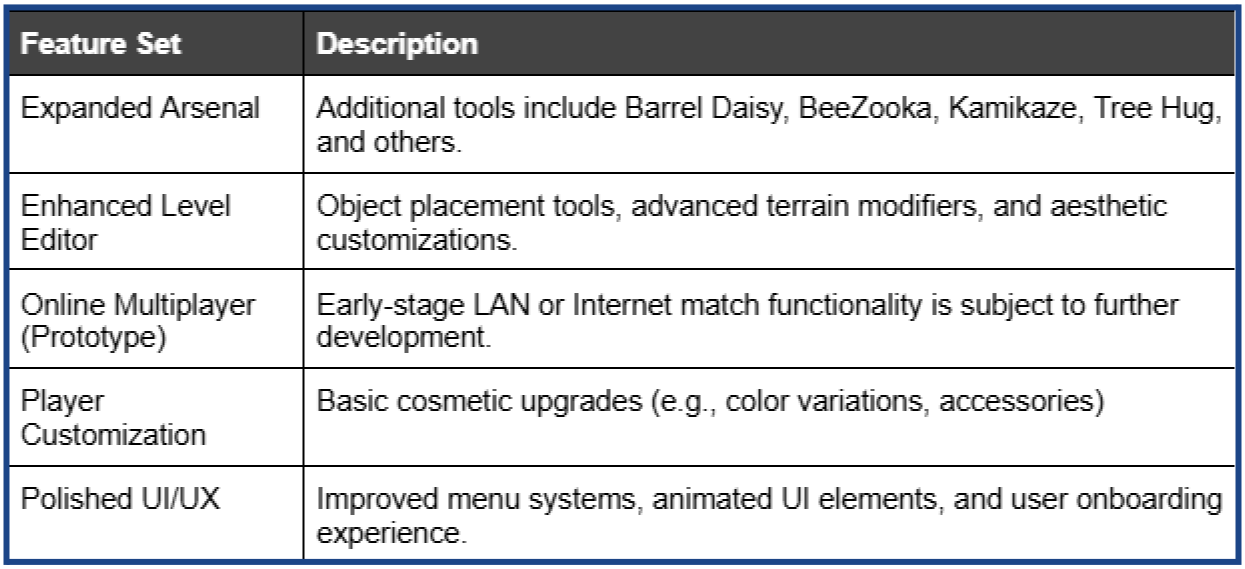
Antillery Beta represents the Minimum Viable Product (MVP) developed during the capstone period. This release delivers core gameplay systems, essential content, and a modular foundation for future expansions. The Beta release comprises local multiplayer functionality, basic player mod support via a bitmap-based level editor, and a curated War and Peace tool arsenal.



Feature Set	Description
Core Systems	Turn-based artillery mechanics, horizontal movement, projectile physics, and destructible terrain.
Arsenal	3 War tools (Bazooka, Grenade, TNT) and 3 Peace tools (Sod Roll, Doom Bloom, Luster Bomb)
Local Multiplayer	Couch co-op for 2-4 players on the same machine.
Terrain System	2D bitmap floating island with destructible and restorable terrain.
Hazard System	Water/lava zone beneath islands for elimination conditions.
UI Elements	Minimalist HUD with weapons menu, power meter, wind indicator, and turn timer.
Visuals	Simple 3D ant models are viewed in 2D orthograpy, with static 2D backdrops and basic animations.
Player Mod Support	In-game bitmap-based level editor allows players to create, save, and load custom maps locally.
Development Tools	Unity Engine (LTS), C# scripting, GitHub VCS, internal debugging utilities.

19.2 Antillery 1.0 (Post-Graduation Expansion)

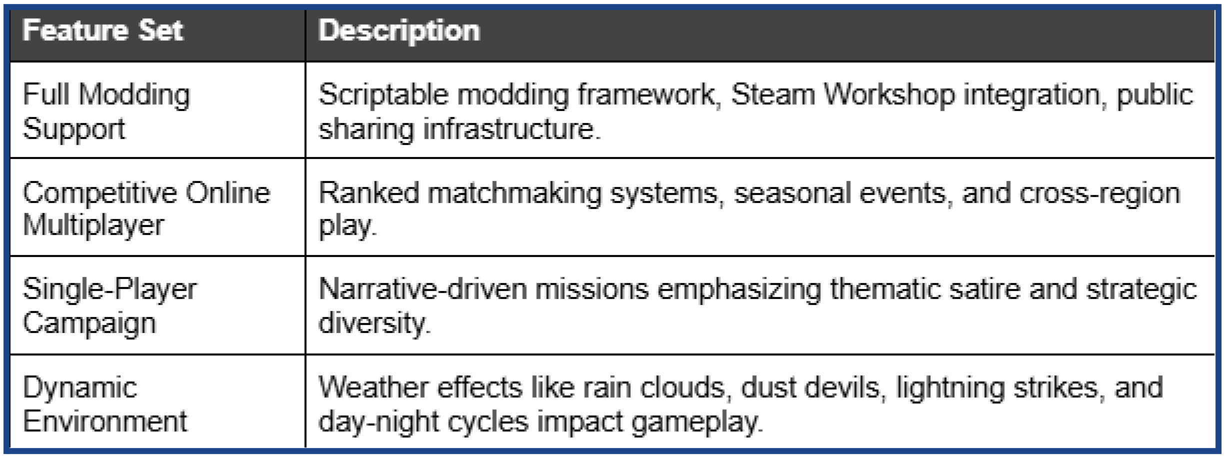
Antillery 1.0 will formalize the first public version, building on the Beta foundation. This version will introduce expanded arsenals, enhanced player experience features, and initial steps toward broader connectivity. These additions reflect impractical improvements during the capstone timeframe but are critical for full public release ambitions.



Feature Set	Description
Expanded Arsenal	Additional tools include Barrel Daisy, BeeZooka, Kamikaze, Tree Hug, and others.
Enhanced Level Editor	Object placement tools, advanced terrain modifiers, and aesthetic customizations.
Online Multiplayer (Prototype)	Early-stage LAN or Internet match functionality is subject to further development.
Player Customization	Basic cosmetic upgrades (e.g., color variations, accessories)
Polished UI/UX	Improved menu systems, animated UI elements, and user onboarding experience.

19.3 Antillery 2.0+ (Long-Term Vision)

Future major versions of Antillery will seek to fully realize the project's expanded vision by integrating robust online infrastructure, comprehensive community-driven modding support, narrative single-player experiences, and advanced environmental dynamics.



Feature Set	Description
Full Modding Support	Scriptable modding framework, Steam Workshop integration, public sharing infrastructure.
Competitive Online Multiplayer	Ranked matchmaking systems, seasonal events, and cross-region play.
Single-Player Campaign	Narrative-driven missions emphasizing thematic satire and strategic diversity.
Dynamic Environment	Weather effects like rain clouds, dust devils, lightning strikes, and day-night cycles impact gameplay.

20 Conclusion

Antillery is a 2D artillery game developed in Unity that reinterprets the genre through nontraditional mechanics and thematic contrast. It draws inspiration from titles like Worms while introducing new design priorities centered on peace tools, environmental dynamics, and player-driven tension. The project delivers a local multiplayer experience shaped by modular design, a clearly defined scope, and realistic development goals.

This document presents Antillery's gameplay systems, technical framework, design intent, and project timeline supporting a complete and feasible capstone. It also establishes the project's purpose within a genre that has seen limited recent evolution, supported by research and comparative data. Mitigation strategies and sprint planning provide a clear path toward delivering a stable, playable beta by the end of the capstone timeline.

In addition to meeting academic requirements, Antillery aims to demonstrate thoughtful game design within real-world constraints. Its completion will contribute a finished, original title to the developer's portfolio. The proposal concludes with thanks for the reader's time.

Grammarly Disagreements

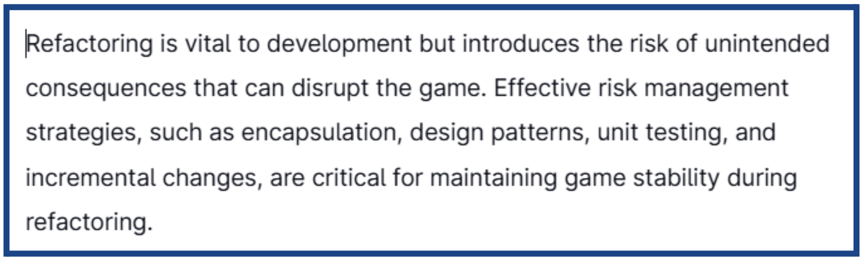
Disclaimer

Grammarly's model and platform can change over time, which may affect the suggestions provided. This document has been revised based on Grammarly's suggestions at the time of submission; any remaining unaccepted suggestions flagged during grading are likely due to updates in the tool's version, algorithm, or platform rather than being genuinely unaccepted. Additionally, the Grammarly Chrome extension no longer supports setting writing goals (academic, report, and MLA), which can complicate conformity with guidelines. This limitation necessitates manually transferring text to the Grammarly site for suggestions, leading to inefficient processes as users must retype changes instead of simply copying and pasting.

Major Grammarly Flaws

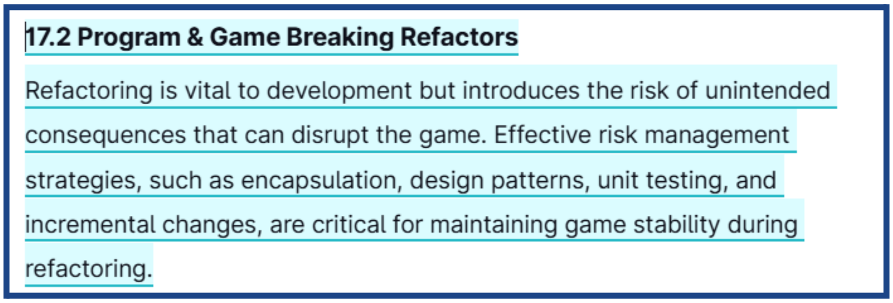
Headers Causing False Positives

- Grammarly deems the following text with no headline as **0% AI text detection**:



Refactoring is vital to development but introduces the risk of unintended consequences that can disrupt the game. Effective risk management strategies, such as encapsulation, design patterns, unit testing, and incremental changes, are critical for maintaining game stability during refactoring.

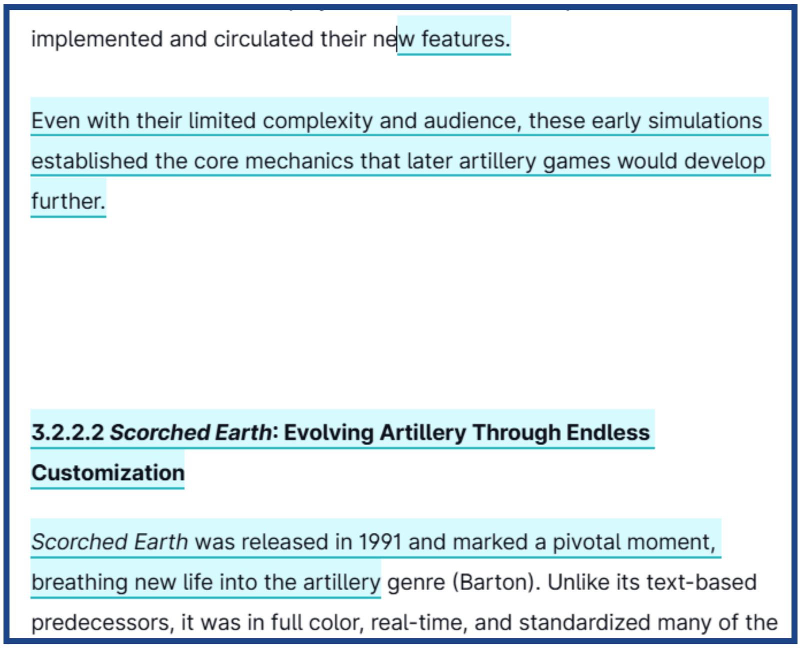
- Adding a headline to the text, Grammarly suddenly detects **100% AI text detection**:



17.2 Program & Game Breaking Refactors

Refactoring is vital to development but introduces the risk of unintended consequences that can disrupt the game. Effective risk management strategies, such as encapsulation, design patterns, unit testing, and incremental changes, are critical for maintaining game stability during refactoring.

Additionally, many of Grammarly's flags for AI detection span from one section over a header and into the next section. These sections are false positives often triggered by the header between sections and not the content in the sections. This point also relates to the example shown in the image above.



implemented and circulated their new features.

Even with their limited complexity and audience, these early simulations established the core mechanics that later artillery games would develop further.

3.2.2.2 *Scorched Earth*: Evolving Artillery Through Endless Customization

Scorched Earth was released in 1991 and marked a pivotal moment, breathing new life into the artillery genre (Barton). Unlike its text-based predecessors, it was in full color, real-time, and standardized many of the

Grammarly Score Changes Randomly

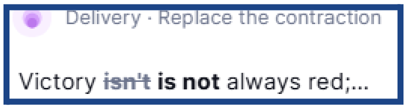
While viewing the document in Grammarly, the software initially calculated a score below 100 despite no outstanding suggestions. After navigating to the plagiarism tab and scrolling through the document, the platform unexpectedly redirected back to the suggestions tab. It updated the score to 100 without any changes to the text. Grammarly's scoring system may refresh or re-evaluate itself based on interface activity rather than content edits.
Video proof:
https://drive.google.com/file/d/1pDgDOIkNtfjeeq4tLf1SLxzgpaHXPyER/view?usp=sharing

No Header 3+ Support

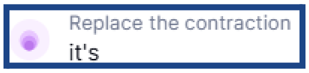
Grammarly's document structure support is limited to two levels of headings (Header 1 and Header 2). However, the project guidelines require a deeper, multi-level hierarchy to organize sections and subsections (e.g., 11.1.3, 14.2.1). As a result, Grammarly does not recognize or evaluate the whole document structure correctly. This lack of support leads to misleading suggestions or formatting flags, particularly in nested sections that are correctly formatted according to MLA and project requirements but fall outside Grammarly's supported scope.

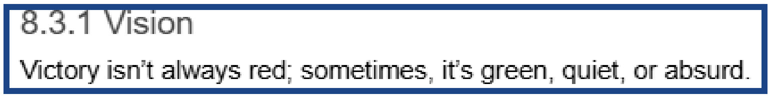
Vision Statement Voice (Section 8)

These suggestions are not accepted because the Vision Statements section is market, investor, and consumer-facing rather than academic and modeled after the standard industry vision statement voice.


Delivery · Replace the contraction

Victory ~~isn't~~ **is not** always red;...

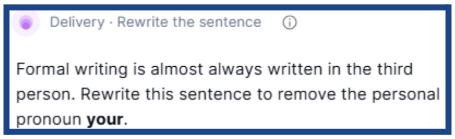

Replace the contraction
it's


8.3.1 Vision

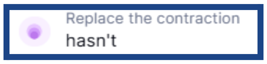
Victory isn't always red; sometimes, it's green, quiet, or absurd.

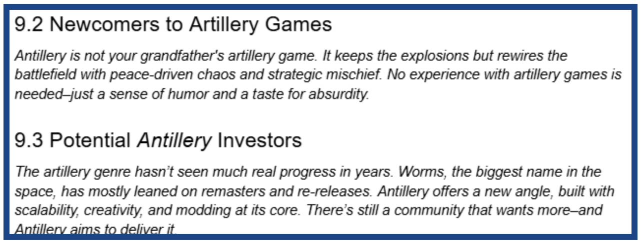
Elevator Pitch Voice (Section 9)

Grammarly flagged tone and style suggestions in the Elevator Pitches section, primarily for second-person pronouns and informal language. These pitches are persuasive and market-facing, not academic.


Delivery · Rewrite the sentence ⓘ

Formal writing is almost always written in the third person. Rewrite this sentence to remove the personal pronoun **your**.

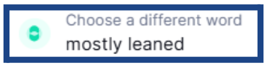

Replace the contraction
hasn't

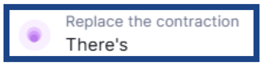

9.2 Newcomers to Artillery Games

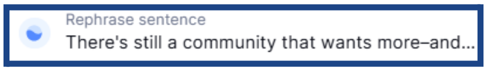
Antillery is not your grandfather's artillery game. It keeps the explosions but rewires the battlefield with peace-driven chaos and strategic mischief. No experience with artillery games is needed–just a sense of humor and a taste for absurdity.

9.3 Potential *Antillery* Investors

The artillery genre hasn't seen much real progress in years. Worms, the biggest name in the space, has mostly leaned on remasters and re-releases. Antillery offers a new angle, built with scalability, creativity, and modding at its core. There's still a community that wants more–and Antillery aims to deliver it.

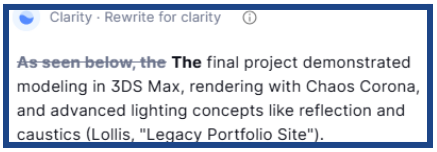

Choose a different word
mostly leaned


Replace the contraction
There's


Rephrase sentence
There's still a community that wants more–and...

"As seen below" (Section 12)

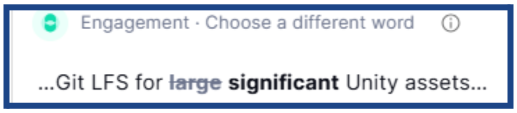
"As seen below" refers directly to the image beneath the text. Removing it would make the reference unclear.


Clarity · Rewrite for clarity ⓘ

~~As seen below, the~~ **The** final project demonstrated modeling in 3DS Max, rendering with Chaos Corona, and advanced lighting concepts like reflection and caustics (Lollis, "Legacy Portfolio Site").

Large =/= Significant in All Cases (Section 16)

Grammarly's green "Engagement" suggestions often replace semantically accurate terms with broader or less appropriate synonyms, such as substituting significant for large, even when dimensionality, not importance, is the intended meaning. These suggestions were reviewed and declined to preserve clarity and technical precision.

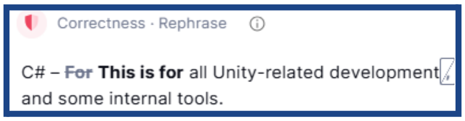


Engagement · Choose a different word

...Git LFS for ~~large~~ **significant** Unity assets...

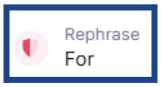
Lists Use MLA-divergent Grammar (Section 16)

Grammarly suggested rephrasing bullet items into complete sentences, which would break consistency. The structure supports skimming and maintains clarity.



Correctness · Rephrase

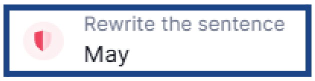
C# – ~~For~~ **This is for** all Unity-related development and some internal tools.



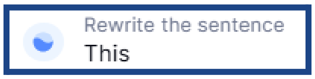
Rephrase
For

Lists Use MLA-divergent Grammar (Section 17)

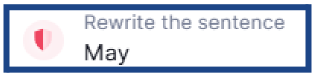
Grammarly suggested rephrasing bullet items into complete sentences, which would break consistency. A list or table supports skimming and maintains clarity.



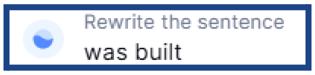
Rewrite the sentence
May



Rewrite the sentence
This

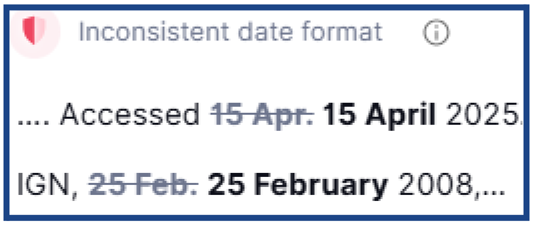


Rewrite the sentence
May



Rewrite the sentence
was built

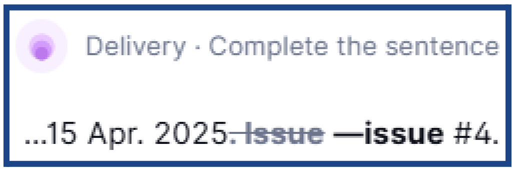
MLA Dates Are Abbreviated



Inconsistent date format

.... Accessed ~~15 Apr.~~ **15 April** 2025

IGN, ~~25 Feb.~~ **25 February** 2008,...



Delivery · Complete the sentence

...15 Apr. 2025~~. Issue~~ **—issue** #4.

Grammarly is making incorrect suggestions about MLA citations.

References

Barton, Matt. "Scorched Parabolas: A History of the Artillery Game." Armchair Arcade, Aug. 2004, https://web.archive.org/web/20071011121637if_/http://www.armchairarcade.com/aamain/content.php?article.51. Accessed 15 Apr. 2025. Issue #4.

Gamma, Erich, et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994

LaFlame, James. "*Worms* Retrospective." IGN, 25 Feb. 2008, https://www.ign.com/articles/2008/02/25/worms-retrospective. Accessed 15 Apr. 2025.

Scacchi, Walt. "Computer Game Mods, Modders, Modding, and the Mod Scene." First Monday, vol. 15, no. 5, May 2010. DOI.org (Crossref), https://doi.org/10.5210/fm.v15i5.2965. Accessed 22 April 2025

Team17. *Worms* Instruction Manual. Ocean Software, 1995. Internet Archive, https://archive.org/details/wormsgamemanual/page/n1/mode/2up. Accessed 23 Apr. 2025.

Team17. "Worms World Party Remastered Web Banner." *Team17*, www.team17.com/games/worms-world-party. Accessed 25 Apr. 2025.

Nuttall, Andy. "War of the *Worms*." The One Amiga, no. 77, Mar. 1995, pp. 25–27. Internet Archive, https://archive.org/details/theone-magazine-77/mode/1up. Accessed 23 Apr. 2025.

Gregory, Jason. Game Engine Architecture. Boca Raton, Fl, CRC Press, Taylor & Francis Group, 2014.

Lollis, Josh. "Portfolio Project Site." *lollisjosh.github.io*, https://lollisjosh.github.io/. Accessed 29 Apr. 2025.

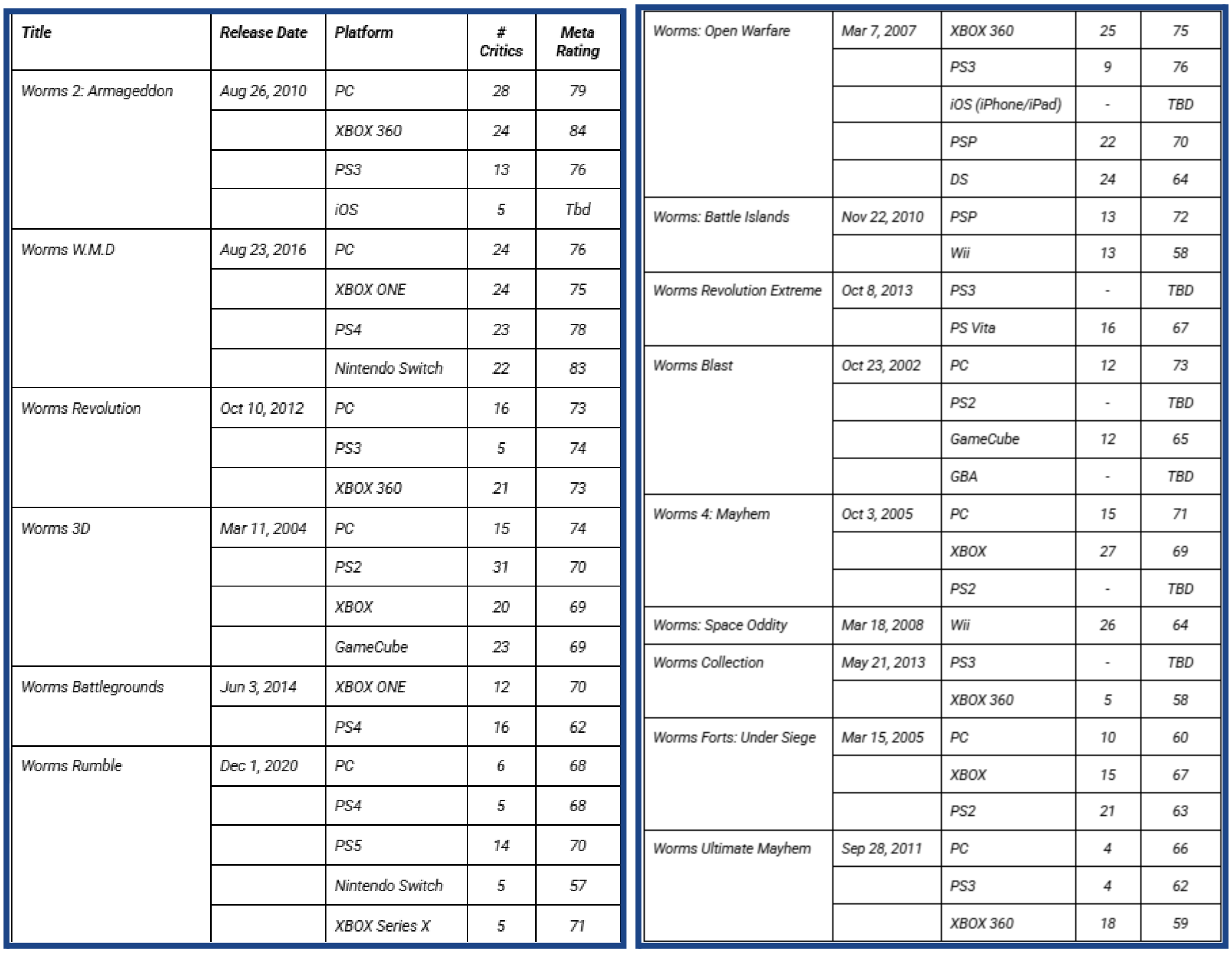
Lollis, Josh. "Legacy Portfolio Site." *Tello-Vision*, https://telloviz.netlify.app/. Accessed 29 Apr. 2025.

Artillery Duel. MicroSparc, 1983. MobyGames, https://www.mobygames.com/game/137141/artillery-duel/. Accessed 12 May 2025.

Appendices

Metacritic Scores for Worms Series (2001–2020)

This table includes all available Metacritic scores for Worms releases.
Link:https://docs.google.com/spreadsheets/d/1atKPqmEiLSnlkTliKoctSYsKvWSu1USG4nKdmxS9VOc/edit?usp=sharing



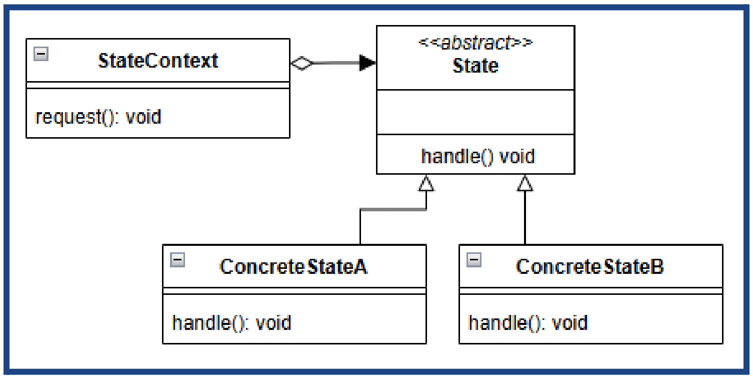
Title	Release Date	Platform	# Critics	Meta Rating
Worms 2: Armageddon	Aug 26, 2010	PC	28	79
		XBOX 360	24	84
		PS3	13	76
		iOS	5	Tbd
Worms W.M.D	Aug 23, 2016	PC	24	76
		XBOX ONE	24	75
		PS4	23	78
		Nintendo Switch	22	83
Worms Revolution	Oct 10, 2012	PC	16	73
		PS3	5	74
		XBOX 360	21	73
Worms 3D	Mar 11, 2004	PC	15	74
		PS2	31	70
		XBOX	20	69
		GameCube	23	69
Worms Battlegrounds	Jun 3, 2014	XBOX ONE	12	70
		PS4	16	62
Worms Rumble	Dec 1, 2020	PC	6	68
		PS4	5	68
		PS5	14	70
		Nintendo Switch	5	57
		XBOX Series X	5	71
Worms: Open Warfare	Mar 7, 2007	XBOX 360	25	75
		PS3	9	76
		iOS (iPhone/iPad)	-	TBD
		PSP	22	70
		DS	24	64
Worms: Battle Islands	Nov 22, 2010	PSP	13	72
		Wii	13	58
Worms Revolution Extreme	Oct 8, 2013	PS3	-	TBD
		PS Vita	16	67
Worms Blast	Oct 23, 2002	PC	12	73
		PS2	-	TBD
		GameCube	12	65
		GBA	-	TBD
Worms 4: Mayhem	Oct 3, 2005	PC	15	71
		XBOX	27	69
		PS2	-	TBD
Worms: Space Oddity	Mar 18, 2008	Wii	26	64
Worms Collection	May 21, 2013	PS3	-	TBD
		XBOX 360	5	58
Worms Forts: Under Siege	Mar 15, 2005	PC	10	60
		XBOX	15	67
		PS2	21	63
Worms Ultimate Mayhem	Sep 28, 2011	PC	4	66
		PS3	4	62
		XBOX 360	18	59

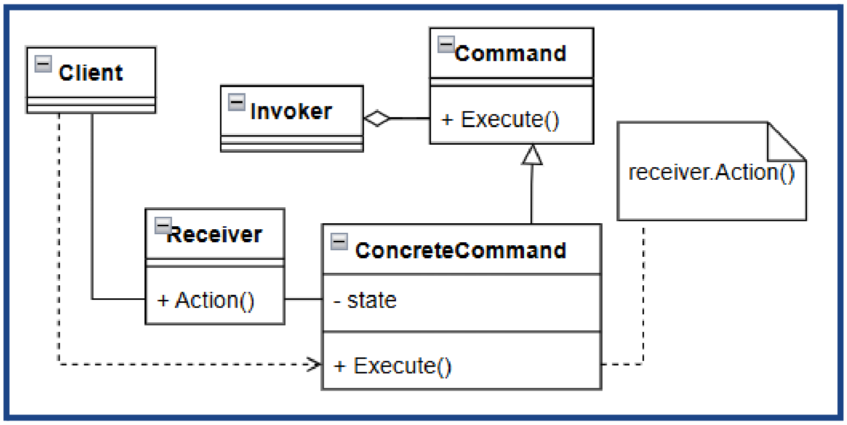
Comparable Mobile Clone Reference (Not Cited)

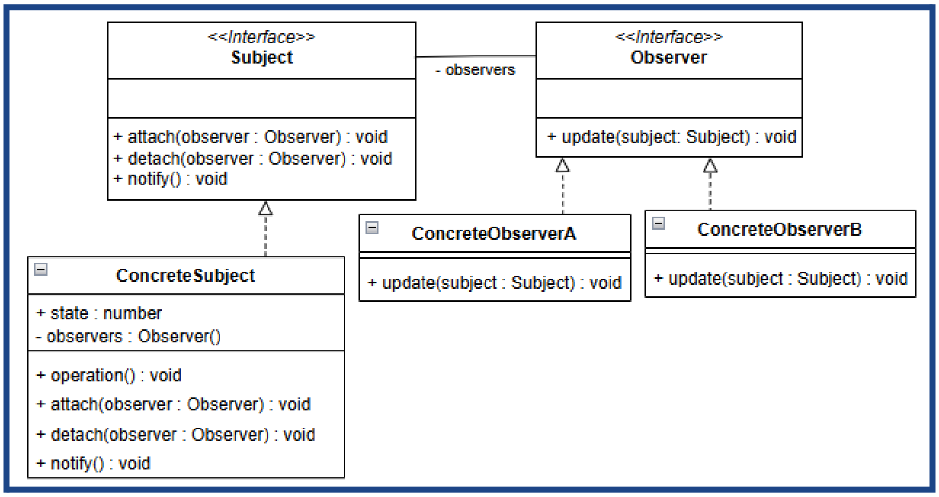
Warlings 2, high-rated Worms-style mobile game.
Link: https://play.google.com/store/apps/details?id=com.warlings5&pcampaignid=web_share
Rating: 4.4/5 based on 67,000+ reviews (Google Play)

Design Pattern Reference (Gang of Four Design Patterns)

Reference diagrams for State, Command, and Observer patterns used in Antillery

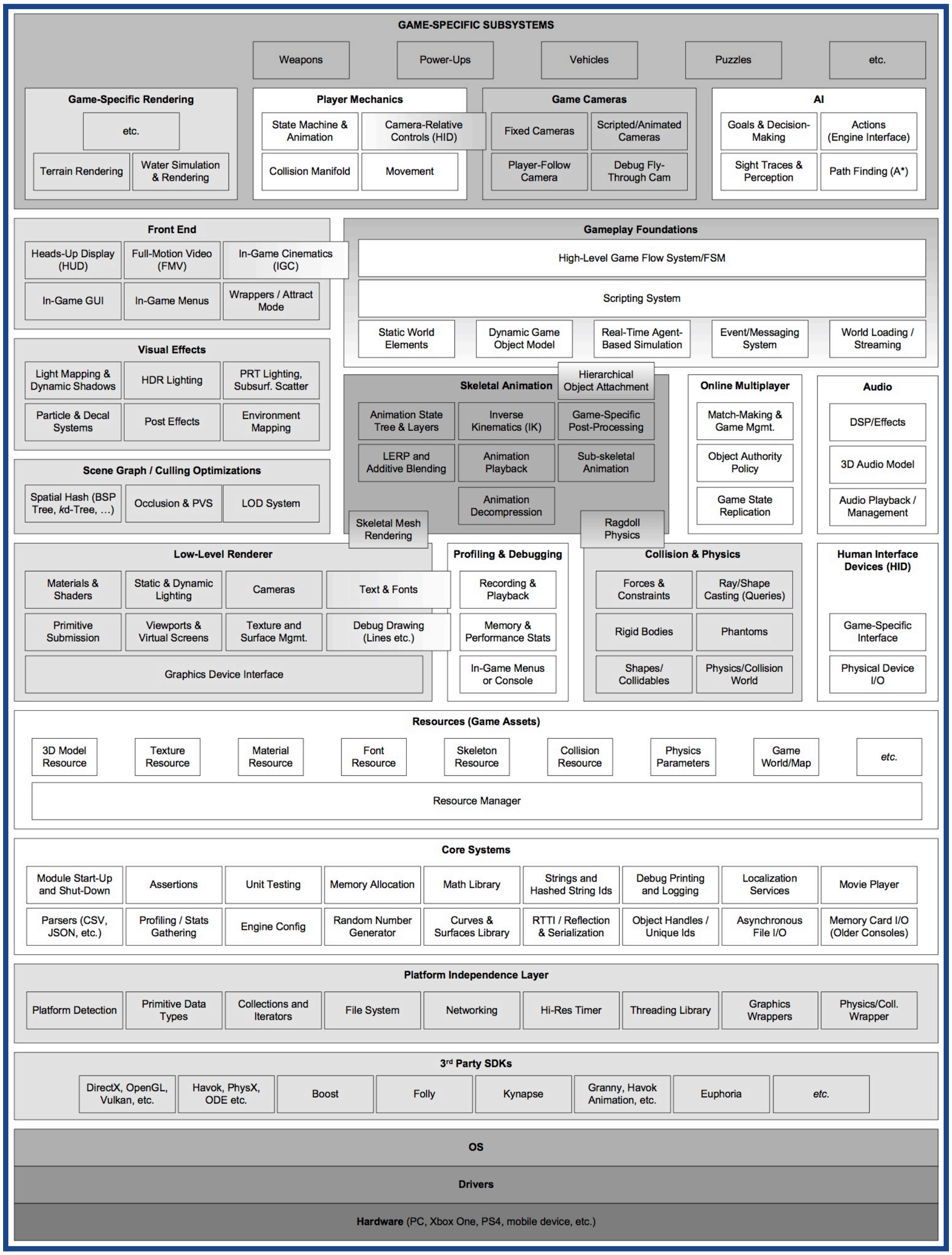





<<Interface>>
Subject
+ attach(observer : Observer) : void
+ detach(observer : Observer) : void
+ notify() : void
- observers
<<Interface>>
Observer
+ update(subject: Subject) : void
ConcreteObserverA
+ update(subject : Subject) : void
ConcreteObserverB
+ update(subject : Subject) : void
ConcreteSubject
+ state : number
- observers : Observer()
+ operation() : void
+ attach(observer : Observer) : void
+ detach(observer : Observer) : void
+ notify() : void

Layered Game Engine Architecture (Source: Gregory)

A game engine stack, as seen in Jason Gregory's Game Engine Architecture. This graph is used in Section 3.1 to contextualize Antillery's system design.


GAME-SPECIFIC SUBSYSTEMS
Weapons
Power-Ups
Vehicles
Puzzles
etc.
Game-Specific Rendering
etc.
Terrain Rendering
Water Simulation & Rendering
Player Mechanics
State Machine & Animation
Camera-Relative Controls (HID)
Collision Manifold
Movement
Game Cameras
Fixed Cameras
Scripted/Animated Cameras
Player-Follow Camera
Debug Fly-Through Cam
AI
Goals & Decision-Making
Actions (Engine Interface)
Sight Traces & Perception
Path Finding (A*)
Front End
Heads-Up Display (HUD)
Full-Motion Video (FMV)
In-Game Cinematics (IGC)
In-Game GUI
In-Game Menus
Wrappers / Attract Mode
Gameplay Foundations
High-Level Game Flow System/FSM
Scripting System
Static World Elements
Dynamic Game Object Model
Real-Time Agent-Based Simulation
Event/Messaging System
World Loading / Streaming
Visual Effects
Light Mapping & Dynamic Shadows
HDR Lighting
PRT Lighting, Subsurf. Scatter
Particle & Decal Systems
Post Effects
Environment Mapping
Skeletal Animation
Hierarchical Object Attachment
Animation State Tree & Layers
Inverse Kinematics (IK)
Game-Specific Post-Processing
LERP and Additive Blending
Animation Playback
Sub-skeletal Animation
Animation Decompression
Skeletal Mesh Rendering
Ragdoll Physics
Online Multiplayer
Match-Making & Game Mgmt.
Object Authority Policy
Game State Replication
Audio
DSP/Effects
3D Audio Model
Audio Playback / Management
Scene Graph / Culling Optimizations
Spatial Hash (BSP Tree, kd-Tree, …)
Occlusion & PVS
LOD System
Low-Level Renderer
Materials & Shaders
Static & Dynamic Lighting
Cameras
Text & Fonts
Primitive Submission
Viewports & Virtual Screens
Texture and Surface Mgmt.
Debug Drawing (Lines etc.)
Graphics Device Interface
Profiling & Debugging
Recording & Playback
Memory & Performance Stats
In-Game Menus or Console
Collision & Physics
Forces & Constraints
Ray/Shape Casting (Queries)
Rigid Bodies
Phantoms
Shapes/ Collidables
Physics/Collision World
Human Interface Devices (HID)
Game-Specific Interface
Physical Device I/O
Resources (Game Assets)
3D Model Resource
Texture Resource
Material Resource
Font Resource
Skeleton Resource
Collision Resource
Physics Parameters
Game World/Map
etc.
Resource Manager
Core Systems
Module Start-Up and Shut-Down
Assertions
Unit Testing
Memory Allocation
Math Library
Strings and Hashed String Ids
Debug Printing and Logging
Localization Services
Movie Player
Parsers (CSV, JSON, etc.)
Profiling / Stats Gathering
Engine Config
Random Number Generator
Curves & Surfaces Library
RTTI / Reflection & Serialization
Object Handles / Unique Ids
Asynchronous File I/O
Memory Card I/O (Older Consoles)
Platform Independence Layer
Platform Detection
Primitive Data Types
Collections and Iterators
File System
Networking
Hi-Res Timer
Threading Library
Graphics Wrappers
Physics/Coll. Wrapper
3rd Party SDKs
DirectX, OpenGL, Vulkan, etc.
Havok, PhysX, ODE etc.
Boost
Folly
Kynapse
Granny, Havok Animation, etc.
Euphoria
etc.
OS
Drivers
Hardware (PC, Xbox One, PS4, mobile device, etc.)