

# Lights Out

## Design Document

Team 37

Client: Josh Deaton

Adviser: Dr. Joseph Zambreno

### Team Members:

---

Tasman Grinnell	Project Manager and Engine Engineer
Joshua Deaton	Rendering Engineer Lead
Lincoln Kness	Game Design Lead
Ben Johnson	System Engineer
Zach Rapoza	Prototyping Engineer
Spencer Thiele	Game Designer
Cory Roth	Game Designer and Rendering Engineer

---

Team Email: [sdmay25-37@iastate.edu](mailto:sdmay25-37@iastate.edu)

Team Website: <https://sdmay25-37.sd.ece.iastate.edu/>

Revised: 5/04/2025 Version 2.0

# Executive Summary

Most mainstream video games focus on conventionally generated Euclidean worlds. Non-Euclidean geometry in many current games either mimics the geometry or is only implemented as research demonstrations, resulting in unenjoyable or flawed gameplay. This project aims to create an engaging game supported by a custom rendering and game engine for accurate Non-Euclidean geometry. Despite the critical need for performance in game engines, current projects involving these unconventional spaces are significantly lacking. The value of the project arises from its ability to expand the boundaries of conventional game visuals and its open-source nature that allows games to flourish in these rarely used geometries.

Some key requirements of this project:

- The game engine must be custom.
- The engine must render Non-Euclidean geometry with custom shaders.
- Standard games built with this engine must render at least 30 frames per second.
- Game mechanics must be interesting and engaging for the user.
- The gameplay must be smooth.

The design described in this document defines our game engine, which supports the requirements specified. Individual submodules support operation by managing independent abstractions (e.g., graphics, sound, input). The engine operates on a loop, handling user inputs, determining what data needs to be rendered, transforming it into the Non-Euclidean space, and then projecting the data back to the user on a display screen. This engine was developed in C++ with various third-party libraries and OpenGL. The targeted hardware is a standard (non-gaming/low-power laptop). The game has been designed to show off this engine called Lights Out. It is a farming simulator that takes on a horror aspect using lighting mechanics and a warping, Non-Euclidean world.

The current state of this project consists of an engine that renders Non-Euclidean math in a {4,5} tessellation format. The currently implemented submodules of the engine include Input Handling, System Scheduling, Rendering Loop, ENT entity-component-system integration, Non-Euclidean Shaders, custom tile maps, Catch2 and CTest unit test integration, and other components necessary to build a game. A fleshed-out game concept and prototype with different mechanics and visuals exists. Currently, only a portion of the designed game is integrated and runs well on the engine and system.

The next steps for the project are to continue integrating more sections and mechanics of the game into the engine, such as farming and lighting. Another future task would be to refine the management of core functionality through performance optimizations. In general, the game engine is working with core functionality to allow the game to be developed. However, further refinement is necessary to enhance the developer and user experience due to the informally defined API the engine currently acts as. With the integration of the Unity-developed prototype, it is clear that additional enhancements are needed to allow a more intuitive development lifecycle and enhance general functionality. By continuing the development of the engine itself, higher performance can be gained for the user, and a more straightforward API can be created to reflect other coding practices prevalent in industry engines.

# Learning Summary

## Development Standards & Practices Used

### Software Practices

- Version Control
- Separation of Concerns
- Software Testing
- User Stories
- Documentation

### Engineering Standards

- Software Life Cycles
- Software Testing
- Project Management - 16326-2009
- Quality Assurance - IEEE 730.1-1995
- Standard for Video Games Vocabulary
- IEEE Standard Glossary of Computer Graphics Terminology
- Information Technology — Computer Graphics

## Summary of Requirements

- Renders Non-Euclidean Geometry
- Run at 30 Frames per Second
- Must be Usable on a Personal Laptop
- Interesting Game Mechanics
- Smooth Gameplay
- Runs on Engine
- Clear External Documentation
- Usable API for Other Projects
- Easy to Install
- Intuitive UI
- Easy-to-Use Controls
- Interesting and Easy-to-Follow Story
- Visually Appealing Game
- Enjoyable Gameplay Loop

## Applicable Courses from Iowa State University Curriculum

- Com S 327
- Engl 314
- Com S 437
- Coms 336
- Math 265

## New Skills/Knowledge acquired that was not taught in courses

- OpenGL
- Unity
- Project Scoping
- Team Interactions
- Resource Management
- Interaction With Clients
- Project Management
- Game Design
- Non-Euclidean Math
- Time Management

# Table of Contents

<b>1 INTRODUCTION .....</b>	<b>8</b>
<b>1.1 PROBLEM STATEMENT .....</b>	<b>8</b>
<b>1.2 INTENDED USERS.....</b>	<b>8</b>
<b>2 REQUIREMENTS, CONSTRAINTS, AND STANDARDS .....</b>	<b>9</b>
<b>2.1 REQUIREMENTS &amp; CONSTRAINTS.....</b>	<b>9</b>
GAME ENGINE.....	9
GAME DESIGN.....	9
OTHER REQUIREMENTS.....	10
<b>2.2 ENGINEERING STANDARDS .....</b>	<b>10</b>
STANDARD 1 .....	11
STANDARD 2 .....	11
STANDARD 3.....	11
ANALYSIS OF STANDARDS .....	11
<b>3 PROJECT PLAN .....</b>	<b>13</b>
<b>3.1 PROJECT MANAGEMENT/TRACKING PROCEDURES.....</b>	<b>13</b>
<b>3.2 TASK DECOMPOSITION .....</b>	<b>14</b>
<b>3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA.....</b>	<b>15</b>
GAME DESIGN TEAM MILESTONES.....	15
GAME ENGINE TEAM MILESTONES.....	15
<b>3.4 PROJECT TIMELINE/SCHEDULE .....</b>	<b>16</b>
GANTT CHARTS.....	16
<b>3.5 RISKS AND RISK MANAGEMENT/MITIGATION .....</b>	<b>19</b>
GAME DESIGN RISKS .....	19
GAME DESIGN RISK MITIGATION PLAN.....	20
GAME DESIGN RISK ANALYSIS.....	21
GAME ENGINE RISKS .....	21
GAME ENGINE RISK MITIGATIONS PLAN .....	22
GAME ENGINE RISK ANALYSIS.....	23
<b>3.6 PERSONNEL EFFORT REQUIREMENTS .....</b>	<b>23</b>
GAME DESIGN HOURS .....	23
GAME ENGINE HOURS .....	25
<b>3.7 OTHER RESOURCE REQUIREMENTS.....</b>	<b>27</b>
<b>4 DESIGN.....</b>	<b>29</b>
<b>4.1 DESIGN CONTEXT .....</b>	<b>29</b>

4.1.1	BROADER CONTEXT.....	29
4.1.2	PRIOR WORK/SOLUTIONS .....	31
4.1.3	TECHNICAL COMPLEXITY .....	32
4.2	DESIGN EXPLORATION .....	33
4.2.1	DESIGN DECISIONS .....	33
4.2.2	IDEATION .....	35
4.2.3	DECISION-MAKING AND TRADE-OFF .....	36
4.3	PROPOSED DESIGN .....	39
4.3.1	OVERVIEW .....	39
4.3.2	DETAILED DESIGN AND VISUAL(S) .....	41
4.3.3	FUNCTIONALITY .....	42
4.3.4	AREAS OF CONCERN AND DEVELOPMENT .....	43
4.4	TECHNOLOGY CONSIDERATIONS .....	43
5	TESTING .....	45
5.1	UNIT TESTING.....	45
5.2	INTERFACE TESTING .....	45
5.3	INTEGRATION TESTING .....	46
5.4	SYSTEM TESTING.....	46
5.5	REGRESSION TESTING.....	46
5.6	ACCEPTANCE TESTING .....	47
5.7	USER TESTING .....	47
5.8	RESULTS .....	47
6	IMPLEMENTATION .....	49
6.1	GAME DESIGN IMPLEMENTATIONS .....	49
	PROTOTYPING.....	49
	INTEGRATION.....	49
	SCENE DEVELOPMENT.....	49
	CURRENT STATUS.....	51
	MONSTERS.....	52
	SHOPS.....	52
6.2	ENGINE IMPLEMENTATIONS .....	54
	CURRENT STATUS.....	54
	MATH .....	56
	EXAMPLE PROTOTYPES .....	57
	NON-EUCLIDEAN RENDERING ITERATIONS.....	58
6.3	DESIGN ANALYSIS .....	64
	FUNCTIONALITY THAT WORKS WELL.....	64
	FUNCTIONALITY THAT DOES NOT WORK AS EXPECTED.....	65
7	ETHICS AND PROFESSIONAL RESPONSIBILITY .....	67

7.1 AREAS OF PROFESSIONAL RESPONSIBILITY/CODES OF ETHICS .....	67
7.2 FOUR PRINCIPLES .....	70
7.3 VIRTUES.....	71
THREE VIRTUES ESSENTIAL TO THE TEAM .....	71
INDIVIDUAL VIRTUES .....	71
 8 CONCLUSIONS .....	 75
8.1 SUMMARY OF PROGRESS .....	75
8.2 VALUE PROVIDED .....	76
8.3 NEXT STEPS.....	77
SUMMARY OF NEXT STEPS.....	77
 9 REFERENCES.....	 79
 10 APPENDICES .....	 81
APPENDIX 1 – OPERATION MANUAL .....	81
ENGINE MANUAL .....	81
GAME MANUAL.....	85
APPENDIX 2 – ALTERNATIVE/INITIAL VERSION OF DESIGN .....	87
INITIAL ENGINE DESIGN .....	87
INITIAL GAME DESIGN .....	87
APPENDIX 3 – OTHER CONSIDERATIONS.....	89
A: PERSONAS AND EMPATHY MAPS.....	89
APPENDIX 4 – CODE .....	93
NON-EUCLIDEAN ENGINE SUBMODULES .....	93
EXTERNAL LIBRARIES .....	95
APPENDIX 5 – TEAM CONTRACT .....	96
TEAM MEMBERS.....	96
REQUIRED SKILL SETS FOR YOUR PROJECT.....	96
SKILL SETS COVERED BY THE TEAM.....	97
PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM .....	97
INDIVIDUAL PROJECT MANAGEMENT ROLES.....	98
TEAM CONTRACT.....	98

# Table of Figures

FIGURE 1 - TASK DECOMPOSITION.....	14
FIGURE 2 - HIGH LEVEL GANTT CHART .....	16
FIGURE 3 - SEMESTER 1 GAME DESIGN .....	16
FIGURE 4 - SEMESTER 1 RENDER ENGINE .....	16
FIGURE 5 - SEMESTER 2 RENDER ENGINE.....	17
FIGURE 6 - SEMESTER 2 RENDER ENGINE.....	17
FIGURE 7 - EXAMPLE POINCARÉ MODEL .....	32
FIGURE 8 - WEIGHTED DECISION MATRIX .....	36
FIGURE 9 - DETAILED DESIGN AND VISUALS.....	41
FIGURE 10 - USER-SYSTEM FEEDBACK LOOP.....	42
FIGURE 11 - HOME SCENE .....	50
FIGURE 12 - FOREST SCENE .....	51
FIGURE 13 - FOREST SCENE WITH MONSTERS .....	52
FIGURE 14 - SHOPPING AREA IN GAME .....	53
FIGURE 15 - FOREST SCENE IN ENGINE .....	55
FIGURE 16 - EQUATION FROM POINCARÉ TO HYPERBOLOID .....	56
FIGURE 17 - EQUATION FROM HYPERBOLOID TO POINCARÉ.....	56
FIGURE 18 - EQUATION DETAILING HYPERBOLIC ROTATION ABOUT X.....	57
FIGURE 19 - SPRITE SHEET RENDER TILES EXAMPLE .....	58
FIGURE 20 - INPUT MANAGEMENT MAPPING SCHEME .....	58
FIGURE 21 - INITIAL NON-EUCLIDEAN RENDERING INTEGRATION TEST .....	59
FIGURE 22 - SECOND INTEGRATION TEST .....	60
FIGURE 23 - INITIAL PQ TILE RENDERING IN SYSTEM .....	61
FIGURE 24 - MULTIPLE PQ TILES BEING RENDERED.....	62
FIGURE 25 - PQ TILES BEING PROPERLY PLACED.....	63
FIGURE 26 - A BUNCH OF CY'S IN A HYPERBOLIC RENDER.....	64
FIGURE 27 - APPENDIX 1: OUTLINE OF ENGINE REPO STRUCTURE .....	81
FIGURE 28 - APPENDIX 1: SUBMODULE CONVENTION.....	82
FIGURE 29 - APPENDIX 1: EXAMPLE INSERTING RESOURCES.....	84
FIGURE 30 - APPENDIX 1: TEXTURE MANAGER INTERFACE.....	84
FIGURE 31 - APPENDIX 1: LOAD TEXTURES EXAMPLE .....	85
FIGURE 32 - APPENDIX 1: USE TEXTURE EXAMPLE .....	85
FIGURE 33 - APPENDIX 1: CONTROLS MAPPING .....	86
FIGURE 34 - APPENDIX 2: REPO STRUCTURE CHANGES.....	87
FIGURE 35 - APPENDIX 2: INITIAL HOME SCENE .....	88
FIGURE 36 - APPENDIX 3: MAX USER PERSONA .....	89
FIGURE 37 - APPENDIX 3: SALLY USER PERSONA.....	90
FIGURE 38 - APPENDIX 3: JORDAN USER PERSONA .....	90
FIGURE 39 - APPENDIX 3: EMPATHY MAP .....	91
FIGURE 40 - APPENDIX 3: PLAYTEST DOCUMENT PAGE 1.....	92
FIGURE 41 - APPENDIX 3: PLAYTEST DOCUMENT PAGE 2 .....	93
FIGURE 42 - APPENDIX 4: CURRENT REPO STRUCTURE .....	94

# Table of Tables

TABLE 1 - GAME ENGINE REQUIREMENTS .....	9
TABLE 2 - GAME DESIGN REQUIREMENTS .....	10
TABLE 3 - OTHER REQUIREMENTS .....	10
TABLE 4 - GAME DESIGN RISKS.....	20
TABLE 5 - RENDER ENGINE RISKS .....	22
TABLE 6 - GAME DESIGN PERSON HOURS .....	25
TABLE 7 - RENDER ENGINE PERSON HOURS .....	27
TABLE 8 - BROADER IMPACT ON AREAS.....	31
TABLE 9 - TECHNOLOGY CONSIDERATIONS.....	43
TABLE 10 - AREAS OF PROFESSIONAL RESPONSIBILITY .....	68
TABLE 11 - FOUR PRINCIPLES.....	70
TABLE 12 - TEAM SKILLSET.....	97
TABLE 13 - TEAM MEMBER ROLES.....	98
TABLE 14 - LEADERSHIP ROLES .....	101
TABLE 15 - MEMBER SKILLS.....	102



# 1 Introduction

## 1.1 PROBLEM STATEMENT

The vast majority of video games in mainstream focus primarily around conventionally generated worlds, using geometry similar to our world. Most Non-Euclidean geometry games are primarily used for research purposes, not general entertainment. Additionally, many of these worlds use computer tricks to pretend they're using Non-Euclidean rendering, even though they use a conventional game development engine. This project aims to create an engaging and enjoyable game with a custom game engine to support the implementation of Non-Euclidean worlds.

Some of the most prominent issues in creating actual Non-Euclidean games involve performance due to the computations required for conversions between spaces. Non-Euclidean spaces do not translate directly, resulting in the need for many calculations and heavy resource costs. Many current projects involving these unconventional spaces have poor performance, and proper management of computational resources is essential.

## 1.2 INTENDED USERS

The people using the final product can come from many different backgrounds. One of these backgrounds would be people who enjoy playing video games. For example, one user's name is Max [Appendix 3 User Personas]. Max is a 17-year-old student. What makes a game enjoyable for him is having ways to grind and optimize his gameplay as much as possible; Max needs a challenge. The proposed solution is a perfect example of something he would enjoy. The plans for this product will allow users to have a goal to grind for through farming and resource gathering, with the added challenges of the world being Non-Euclidean.

Another user is Sally, an artistic 20-year-old [Appendix 3 User Personas]. She needs to have an enjoyable game with an artistic touch because she loves art in all different forms, ranging from music to the artistic style of the in-game sprites. The proposed solution aims to make the artistic experience as good as possible. Meshing sprite art with the Non-Euclidean space will create an interesting distortion on all of the game assets, adding to the horror experience of the proposed solution.

Lastly, the user Jordan [Appendix 3 User Personas] is a 25-year-old ESports Professional. What Jordan wants from a game is a way to have a challenging and engaging gaming experience because he enjoys challenges and becoming the best gamer he can be. An unconventional combat system will add a new, challenging twist to combat enemies throughout. Using different traps and yourself as bait will add an innovative combat system that someone like Jordan would enjoy. All of these will be amplified by the Non-Euclidean geometry, creating a fresh experience for all gamers.

## 2 Requirements, Constraints, And Standards

### 2.1 REQUIREMENTS & CONSTRAINTS

The primary requirements involve many baseline expectations for functional software, primarily with performance and soft design requirements. For the game engine, the primary requirements are to meet the requirements specified by the game design team to support the game's operation in general. The requirements are divided into various sectors: functional, physical, and user experience.

#### Game Engine

Type of Requirement	Requirement
Functional	Renders Non-Euclidean geometry
Functional	Renders vectors of data into a laptop
Functional	Runs at 30 fps ( <b>constraint</b> )
Resource	Not exceed that of a personal laptop
Physical	A graphics processing unit is required on a laptop
Aesthetic	The code should be readable
Aesthetic	The code should be commented
User experiential	A good interface will be provided with good documentation
Interface	Be consistent with other render engine interfaces

*Table 1 - Game Engine Requirements*

#### Game Design

Type of Requirement	Requirement
Functional	Easy to understand, yet interesting game mechanics
Functional	Smooth gameplay
Functional	Run on the render engine
Resource	Internet connection to install
Resource	Uses the game engine
Physical	A personal laptop with some form of GPU

Physical	Some form of user input (Keyboard/Controller)
Aesthetic	Pleasing Art Style
Aesthetic	Pleasing music/sounds that mesh well
User experiential	Intuitive UI
User experiential	Easy-to-use controls
User experiential	Easy-to-follow story
Economic/Market	Easy to monitor and control the distribution
Economic/Market	Demand for cool games
Interface	Keyboard & mouse/controller compatibility

*Table 2 - Game Design Requirements*

### Other requirements

Type of Requirement	Requirement
Timing	Some working version of the game by the end of CPRE 4920
Timing	A final version of the engine by the end of CPRE 4920
Installation	Easy to install

*Table 3 - Other Requirements*

It can be seen from these tables that there is a wide range of requirements for this project. The main functional requirements consist of making a working solution. The only hard constraints provided are that the game engine must render Non-Euclidean math and that the engine must run at a reasonable framerate. The rest of these requirements come from the standard of work that should be created. This project does not aim to create a bare-bones project but a fully fleshed-out idea. The aesthetic and user experiential requirements will set this project apart from others in the same field.

## 2.2 ENGINEERING STANDARDS

Engineering standards are essential because they are the guiding principles for developing a complete solution. Standards are the guiding force for products to be safe, reliable, and consistent. Engineers can meet the expectations during development while establishing guidelines and good practices. Users would not have any expectations of what to expect from companies without standards. Also, there would be a significant disparity between various companies and components. Standards promote uniformity, which allows for ease of use by the consumer.

## Standard 1

Standard for Video Games Vocabulary [1]

This standard goal is to unify the vocabulary used in the video game industry. It aims to remove ambiguity in terms of the use of artificial intelligence that can be used within industry. It precedes how characters speak and how industry professionals develop new material. Terms will be defined based on existing literature and the community's consensus. This standard is relevant because the proposed solution to this project is a video game, so the solution should adhere to those standards.

## Standard 2

Quality Assurance - IEEE 730.1-1995 [2]

This standard's goal is to put in place good practices when it comes to the development and maintenance of software. It is a standard focusing on where failures could occur and how failures could impact the safety of its users. It makes sure that plans are in place in case of failure. This is relevant to the project as a plan will be implemented if the game engine fails when a user uses it. The same idea also applies to the proposed solution.

## Standard 3

Software cycles [3]

This standard goal establishes a common framework for the software life cycle. It also defines terminology that the industry should use when referencing software life cycles. It aims to provide reasonable standards for developing software regarding the timeline. It also wants to provide context for software products regarding their development. This international standard also provides processes to help define and improve existing life cycle systems in organizations. This is an essential standard for this project. A solid framework is critical for this project's success and how the software lives.

## Analysis of Standards

The three standards above seem relevant to this project, but may not be the most pertinent. The first standard is of utmost importance. The proposed solution is a video game; the standard should be followed to produce a high-quality video game. While documenting how the game works and implementing communication in the game, the vocabulary standard defined by this standard must be followed. This standard may not be relevant to the main structure of the project, but it is appropriate when it comes to the quality of the product produced.

The second standard may have less of an impact on the project overall. While it is essential to have a defined plan for failure to help the users when our software fails, the software being developed is low risk. The project will not put users in high-risk environments or build something that handles sensitive information. The standard is essential, but it should not be the first consideration.

Standard three is vital in the later stages of this project and will become more important if this project continues outside of senior design. Setting up a framework for the software life cycle is not a priority if you don't have working software. For this to be important, the software needs to be working. The software being developed is not planned to work until the 2nd semester, so this is not

a consideration until then. However, a good framework is vital to maintain the game appropriately once there is working software.

Another standard looked at was Software Testing, which will be considered once the testing stage of the project begins. Standards revolving around rendering were looked into to see if there were any standards on how to use/create/maintain an engine in software like OpenGL. Because an interface for this engine was a requirement, standards regarding computer graphics and computer graphics interfaces were researched.

## 3 Project Plan

### 3.1 PROJECT MANAGEMENT/TRACKING PROCEDURES

This project will adopt a hybrid of both waterfall and agile approaches. The overarching method will consist of a waterfall approach for the high-level structure of the timeline and task decomposition, but the specific task completion will use an agile approach. This structure was chosen mainly because of the time constraint given. This timeframe is only 2 semesters to produce the working final deliverables. Due to this constraint, a more rigid schedule is favored to ensure deadlines are met. This rigid schedule better aligns with a waterfall approach.

Another reason this approach was adopted was the dependency of tasks. Specific tasks depend on the previous task being completed, so a more agile approach cannot be taken. Things can only be iteratively improved upon when there is something to be improved upon. Once a semi-working video game prototype exists, a more iterative approach can be taken towards tasks and problems. This means a shift can happen over the year from a heavy focus on waterfall to an increasing emphasis on the agile approach.

Finally, an agile approach to completing tasks was chosen because it makes changes to the requirements easier. As a student-proposed project, the client is a student, which allows for more frequent client feedback. It also allows the project's direction to be changed more seamlessly. This allows for more flexibility with the requirements and improved group decision-making.

GitHub was used for version control and task tracking. Significant tasks and schedules were maintained externally in Google Sheets using Gantt Charts. The significant tasks were tracked this way because these tasks and deadlines will remain relatively stable. Smaller tasks were tracked on GitHub because it was easier to update deadlines and issues when taking more iterative approaches to completing them. Weekly meetings were used to help keep track of approaching deadlines and ensure that the schedule remained on track.

### 3.2 TASK DECOMPOSITION

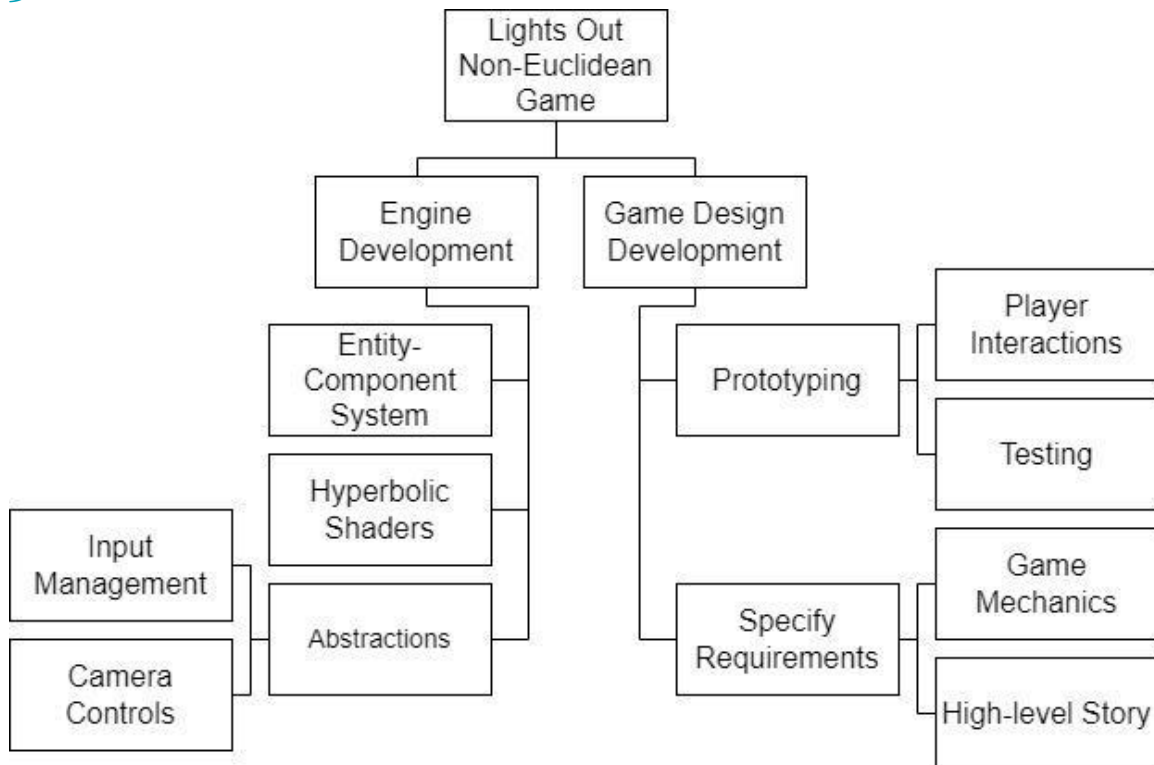


Figure 1 - Task Decomposition

The overall decomposition of tasks involves decomposing the high-level tasks between the Game Engine and Game Design Teams. Each team will address the appropriate tasks to complete the engine and design requirements. Along with this, teams are responsible for ensuring communication between teams, allowing for newly defined design requirements to be communicated to and met by the rendering team.

The game design team will perform the right branch of the task decomposition tree, pictured above, while the engine team will operate on the tasks presented in the left branch. The game design tasks (prototype fundamentals and specifying design requirements) will be discussed and worked on in parallel, with the design requirements communicated to the engine team for planning purposes. Additionally, the prototypes made in Unity will act as proof of concepts for the game itself before implementation on the engine.

The tasks assigned to the engine team revolve around creating an engine that can support the specific requirements of the design team, not including additional or redundant features similar to those found in industry-standard engines such as Unity or Unreal. The underlying tasks are the general implementation of the engine while keeping requirements in mind for creating a straightforward and valuable API that can be used during the game development phase.

### 3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

#### Game Design Team Milestones

- Game Selection
  - Choosing the genre and the overarching theme of the final game design.
- Design Document
  - Once a game has been chosen, the next milestone would be creating an in-depth document detailing the main features of the game, i.e., what core features will be included.
  - Main NPCs, Biomes, Genre.
  - Develop a larger story matching the genre and theme.
- Main Prototypes Creation
  - Implement the primary and core features needed for a minimal viable product demo.
- Single Scene Creation
  - Integrating the core features into a singular scene.
- Functional Demo
  - Getting a demo of multiple scenes, demonstrating the main gameplay features, and the gameplay loop.

#### Game Engine Team Milestones

- Mathematical Modeling
  - Must determine which form of Non-Euclidean math will be implemented.
- Basic Rendering
  - This milestone is being able to render basic sprites, shapes, and features in OpenGL.
- Core Feature Implementation
  - Implement the following core features that are needed in the game:
    - Sprites
    - Entities
    - Lighting
    - Collision
- Math Implementation
  - Ability to render the above in a Non-Euclidean manner.
- Engine Demo
  - Create a basic demo on an example maze map to showcase the engine.
- Running a Video Game on the Engine
  - Ability to run all of the video game scenes created by the game design team



## Gantt Charts

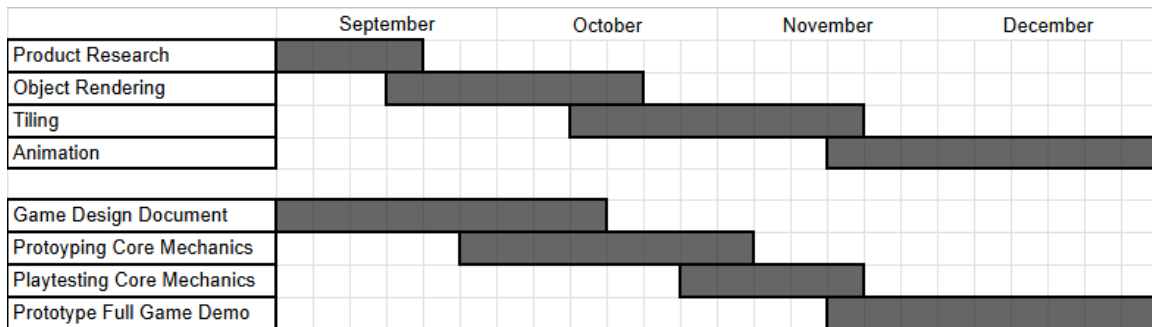


Figure 2 - High Level Gantt Chart

## GANTT CHART Game Design

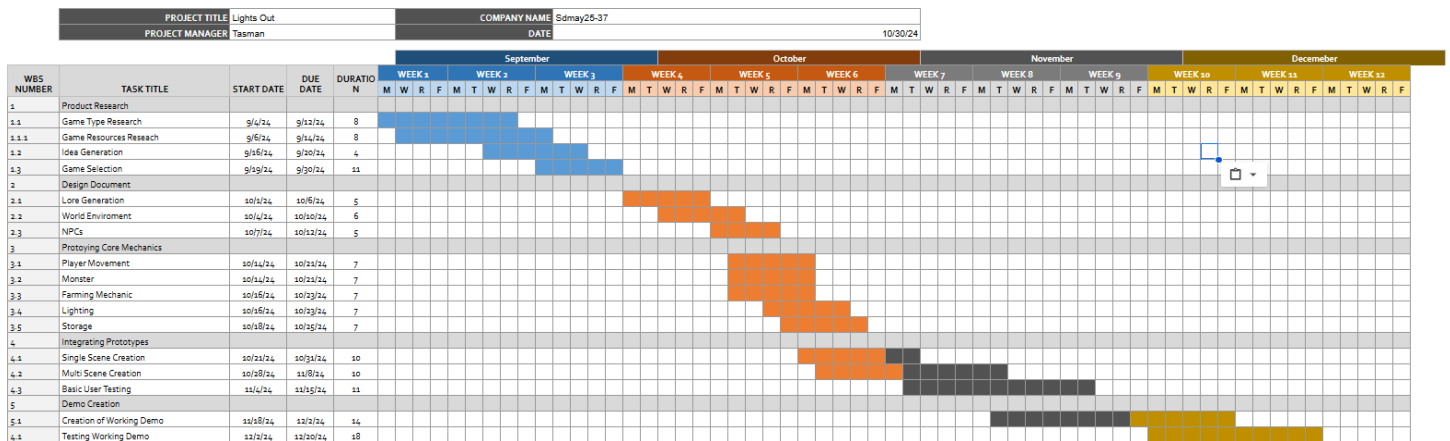


Figure 3 - Semester 1 Game Design

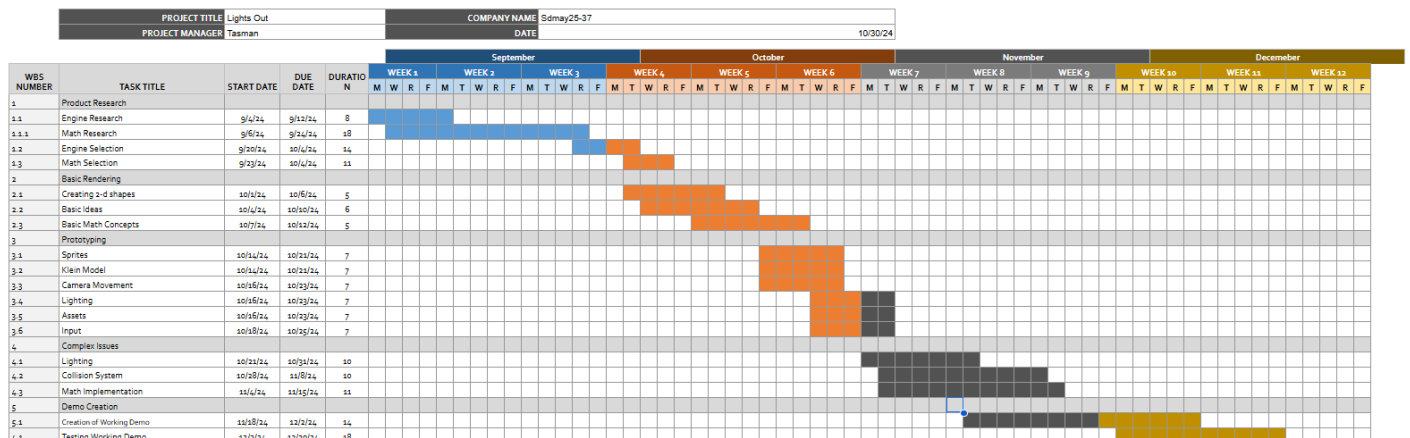


Figure 4 - Semester 1 Render Engine

#### GANTT CHART Render Engine

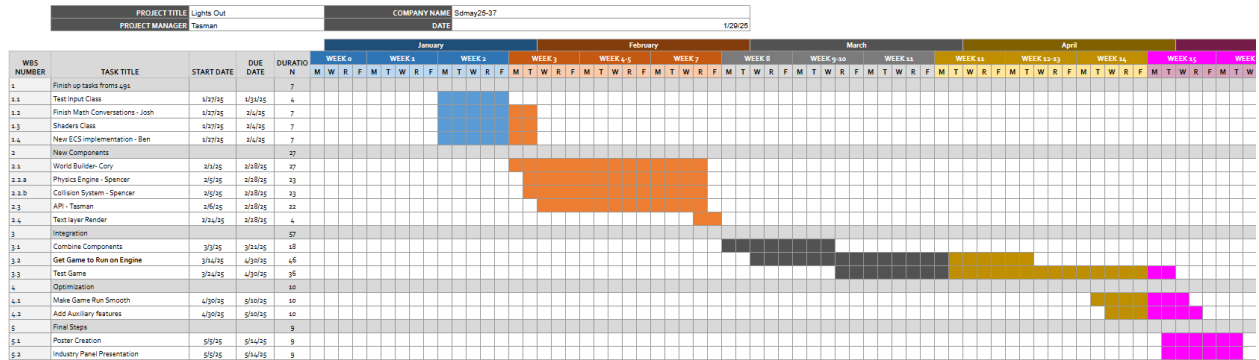


Figure 5 - Semester 2 Render Engine

#### GANTT CHART Game Design

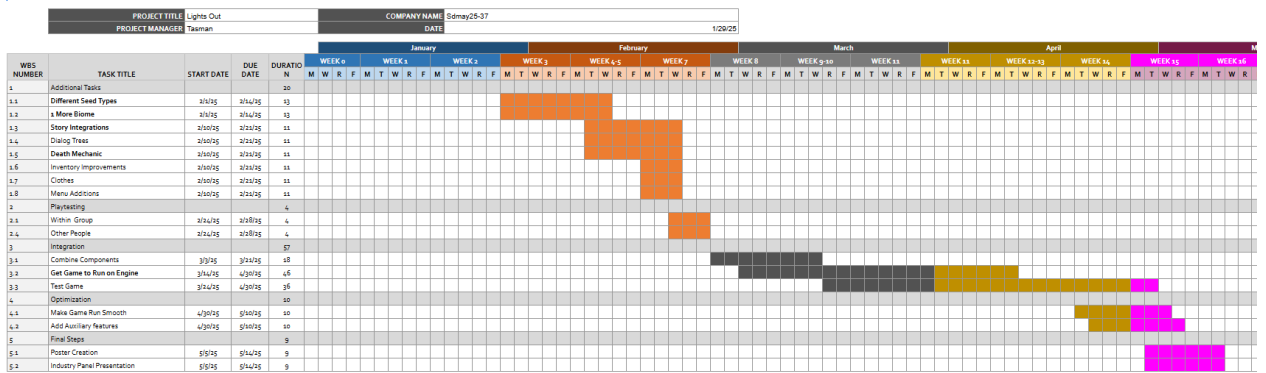


Figure 6 - Semester 2 Render Engine

The Gantt charts made for CPRE 4910 and CPRE 4920 are shown above, detailing the planned schedule of tasks this project will complete and when they should be completed. It is divided into two main groups: Game Design and Game Engine. Communication between groups happens regularly about functionality and specifications. However, it was deemed easier to split the tasks as the game design team will do different tasks than the Game engine team. Both teams started by doing project research. The Game engine will spend more time on the Non-Euclidean math in this part of the project due to its heavier reliance on it. The game design team will primarily research what makes a video game suitable for creating a good solution. Both teams have a part of their schedule with learning the software because most members need to gain experience with these software technologies. Once each team has a reasonable understanding of the software, implementation of the core functionality of the game/engine can begin. These core functionalities are called prototypes because they are separated and can be tested independently. Once prototypes have been created, they will be integrated to make an initial working prototype of what the game will look like/how the game engine will operate.

By the end of semester one, it was estimated that there would be some deliverables of a demo of the game's main mechanics prototype in Unity and that the underlying rendering of Non-Euclidean math would be doable for the engine. These dates were assigned to ensure that in semester 2, enough time would be allocated to integrating the game into the engine. After the first semester,

there was a better understanding of the system design and what efforts were needed to focus on for the second semester. The game's main mechanics demo was met on time, and the rendering engine deadline was met with lower effectiveness than initially thought. This was because the engine lacked support for multiple tiling at that state. Additionally, the team was unable to make as much progress as we were expecting due to understanding the responsibilities and necessary components for the engine itself.

The main tasks of semester 2 were to build the components in our engine that the game needed and start integration. Over the course of the second semester, the engine was finished in a barebones state to allow for game development to occur. The primary blocking factors were due to game design playtesting, issues with the shaders, and dealing with the intricacies of Non-Euclidean math. Although all these issues were dealt with, it set back the timeline of the project, thus reducing the ability to integrate fully., The performance of multiple engine subsystems prevented our functional goals from being reached, which was another blocking factor.

### 3.5 RISKS AND RISK MANAGEMENT/MITIGATION

#### Game Design Risks

Task	Main Risk	Probability	Reason for Probability
Product Research	Misidentify user needs	0.1	Team members are part of the user group and thus know user needs.
Game Type Research	Lack of Quality Research	0.1	Team members are part of the user group and thus know user needs.
Game Resources Research	Using not state of the art tools	0	Have people on the team who know what the state-of-the-art tools are.
Idea Generation	Lack of Innovation in Game Choice	0.6	See Mitigation
Game Selection	Choosing a game with too big a scope	0.7	See Mitigation
Design Document	Bad design, so harder to build later	0.4	Members are passionate about this project, so the effort will be put into place to determine quality.
Lore Generation	Uninteresting/too complex for the user	0.3	Need for the game to be interesting, can take inspiration from other games.
World Environment	Boring Gameplay	0.4	
NPCs	Lack of player immersion	0.4	Taking Inspiration from other games and seeing how people react to other games allows for better decision-making.
Prototyping Core Mechanics	The game will not work properly		
Player Movement	The player will not be able to move	0.1	Basic Feature.
Monster	Inconsistent Monster Behavior	0.2	Feature already implemented.
Farming Mechanic	The game will to tedious	0.2	Experience from other games inspired how this mechanic will feel.
Lighting	Lose Player Interest	0.4	More critical because it is a core mechanic of the game.
Collision	Interactions between objects won't work	0.2	It is crucial, but left to the game engine to figure out.
Storage	Boring Gameplay cycle	0.1	Prior experience will dictate how this is designed.
Integrating Prototypes	Buggy Experience		

Single Scene Creation	Prototypes won't work with each other, leading to a delay	0.4	It is crucial to demo/ explain to an outside person. Integrating multiple people's work is always a challenge.
Multi-Scene Creation	Player Information will not carry over between scenes	0.4	Making more scenes is not as difficult if one scene can be created.
Basic User Testing	The game does not feel enjoyable to play	0.4	Testing to make sure the game feels smooth. Members are not experts in this field, so bugs will happen.
Demo Creation	Delays in previous steps may lead to a lack of time		
Creation of Working Demo	Too big of a scope the project cannot fit everything in. This leads to the project not being done.	0.7	See Mitigation
Testing Working Demo	The game does not work	0.4	Testing to make sure the game feels smooth. Members are not experts in this field, so bugs will happen.

*Table 4 - Game Design Risks*

### Game Design Risk Mitigation Plan

- **Idea Generation and Game Selection:**

This is a high risk for this project because it is an essential aspect. A primary need of users is enjoyment, so the game idea needs to be a well-polished one that can bring a high level of player immersion. This is a risk due to the team's limited experience in game development.

This risk is mitigated by extensively researching other games and taking inspiration from them. This way, this game will not take unnecessary risks by generating an entirely new idea, but instead put a new spin on a genre well-perceived by the users. This leads to risk mitigation because there is proof that this game genre can have high player immersion if done correctly.

- **Creation of Working Demo:**

This will be a high-risk task for this project for many reasons. The first reason is that the schedule may need to catch up on pace. This means the proper amount of time for this semester's tasks will not be achieved. Another reason is that tasks require all of the previous tasks to work correctly and be able to work together. There are integration tasks and testing to mitigate this, but issues are likely to arise still.

The primary way to mitigate this risk is by keeping a hard internal deadline for smaller tasks. There is a need to stay on track with the schedule to have the proper time allocated for this task. If this is not possible, the schedule must be adjusted accordingly. This will result in some tasks more auxiliary to the project being transferred to the second semester to ensure the proper allotment of time for these tasks. A list of resources also exists that can be used to help complete tasks on time and give advice on how to go about completing tasks.

## Game Design Risk Analysis

Due to the mitigation plan detailed above, the team was able to prevent the risk that the game would not be enjoyable. The risk was prevented by extensively researching other games and taking inspiration from them. This way, this game did not take unnecessary risks by generating an entirely new idea, but instead put a new spin on a genre well-perceived by the users. This leads to risk mitigation because there is proof that this game genre can have high player immersion if done correctly. From the results of our user testing, it was concluded that the game idea generated was enjoyable to those who participated.

One risk that caused delays and more time spent on it was the creation of a working demo. When creating a demo, the team ran into many more bugs and issues than expected. One reason was the lack of consistent coding between team members; when integrating different mechanics, these subcomponents did not interface well. This led to more time spent on integration than expected. Another reason was the lack of experience developing a game from scratch, which led to the team underestimating the number of bugs within a system. This slowed down the ability to conduct a meaningful playtest and delayed the start of integration into the engine.

## Game Engine Risks

Task	Main Risk	Probability	Reason for Probability
Product Research	Misidentify user needs	0.1	Team members are part of the user group and thus know user needs
Engine Research	Lack of Quality Research	0.2	There is a limited number of rendering languages.
Math Research	Don't understand the math	0.4	This is complex math, so time is needed to process, understand, and then implement.
Engine Selection	Choosing an engine that has a high learning curve adds delay to the creation of features	0.1	It was already chosen before the project started.
Math Selection	Certain Non-Euclidean spaces are more complex, so math gets more complicated and worse performance	0.2	Limiting number: choose one that would be relatively simple to implement.
Basic Rendering	Don't properly learn to render, so features take longer	0.6	See Mitigation
Creating 2-D shapes	Don't properly learn to render, so features take longer	0.3	There are plenty of tutorials, so it should not be a significant issue.
Basic Ideas	Don't properly learn to render, so features take longer	0.3	There are plenty of tutorials, so it should not be a significant issue.
Basic Math Concepts	Don't understand the math	0.5	See mitigation
Prototyping	Delays core development if stuck on this for too long		

Sprites	Bad rendering and bloated assets	0.2	Bloated assets are not a worry until the optimization of the engine.
Klein Model	The engine won't meet technical specs	0.9	See mitigation
Camera Movement	Jagged Movement makes gameplay less fun	0.4	Tutorials exist, but this might be an issue, given the different environments being built.
Lighting	Lose Player Interest	0.4	A core feature for Game Design needs to work well.
Assets	Bad code management and poor optimization	0.3	Bloated assets are not a worry until the optimization of the engine.
Input	Cannot integrate well with game design	0.2	I/O is well-documented and can be tested easily.
Complex Issues			
Lighting	Loss of performance	0.2	A core feature for Game Design needs to work well.
Collision System	Will not be able to implement game design features	0.4	See mitigation
Math Implementation	The engine will not meet technical specifications	0.9	See mitigation
Demo Creation	Delays in previous steps may lead to a lack of time		
Creation of Working Demo	Too big of scope for the project, cannot fit everything done into the timeframe	0.5	Scoping a project is a new skill to most members, so the chosen idea may be too much to handle.
Testing Working Demo	The engine does not work	0.5	There is no prior experience on the team, so the team does not know what to expect when testing/verifying correctness.

Table 5 - Render Engine Risks

## Game Engine Risk Mitigations Plan

- **Math**

The main risk for the game engine is being able to render Non-Euclidean math. None of the team members are math majors; the computations to convert from a Euclidean space into a Non-Euclidean space are non-trivial. This project is not just learning how to render, but also how to render in a new space. The main risk is that the correct computations/conversions between spaces are not done due to a failing to understand the math adequately.

The primary risk mitigation strategy will be working as a team and sharing knowledge collectively. The approach is to learn this math independently, then come together as a group and compare and contrast what was learned. By having multiple people learn from each other, the risk is mitigated that incorrect information will be learned. A list of

resources that can be used is also created in case advice is needed on how to better this understanding.

### Game Engine Risk Analysis

The main risks experienced for the engine were identified, with performance issues and the shaders serving as blockers throughout much of the project. In the final weeks of the project, the team struggled with optimizing the shaders and performance of the tilemaps, which was one of the most significant issues in the project itself. These issues were incredibly apparent when preparing demos for the final deliverables, with the prototypes we created taking multiple seconds to load.

However, after hosting various live coding sessions, the team was able to make progress and solve the issues that were experienced.

## 3.6 PERSONNEL EFFORT REQUIREMENTS

### Game Design Hours

Task	Estimated Person Hours	Reason	Actual Person Hours	Reason
Product Research	100	Want to spend 2-2.5 weeks coming up with a good idea.	80	From background knowledge had a better understanding than anticipated
Game Type Research	20	Need to determine what users like/dislike. Each member puts in 5 hours, so everyone has an idea.	10	From background knowledge had a better understanding than anticipated
Game Resources Research	15	This can be done relatively quickly, but members should know what resources exist.	10	From background knowledge had a better understanding than anticipated
Idea Generation	40	There should be four people taking ~1 week time to create a good initial plan.	40	
Game Selection	25	~6 hours per member to flesh out ideas and ensure solid ideas exist.	20	



Design Document	85	The game needs to be interesting. Time needs to be spent making it so.	100	Spend more time fleshing out ideas
Lore Generation	25	Lore is important to keep the players invested.	30	
World Environment	30	Exploration is a significant part of game design, which needs an exciting world.	40	
NPCs	30	It is essential to understand why certain games are loved/hated.	40	
Prototyping Core Mechanics		Each core mechanic is being given to one person with 1.5 weeks to implement and test.		Some tasks were easier than expected, whereas others were harder than expected
Player Movement	15	1.5-2 weeks' worth of work for one person.	10	
Monster	15	1.5-2 weeks' worth of work for one person.	25	Different renditions of pathfinding + collision of the player caused it to take longer
Farming Mechanic	15	1.5-2 weeks' worth of work for one person.	25	Different ideas of how to plant/ what to plant
Lighting	15	1.5-2 weeks' worth of work for one person.	10	
Collision	15	1.5-2 weeks' worth of work for one person.	20	2 different ways to go about collision, looked into both options
Storage	15	1.5-2 weeks' worth of work for one person.	10	
Integrating Prototypes	200	Very important, the team should spend a month making a solid game.	170	Less time was spent here as it was grouped into demo

				creation. Once two scenes were created, moved on to demo creation
Single Scene Creation	40	~ 1 week to integrate core mechanics.	40	
Multi-Scene Creation	80	~ 2 weeks to create more scenes and add more mechanics. Allows for documentation and preparation for the second semester.	60	
Basic User Testing	80	~ 2 Weeks. There will be bugs; time is needed to test/ change the implementation based on feedback.	70	
Demo Creation	150	Want to make some polished demos to make a good product Also, building some extra time in case the team gets behind schedule ~3 weeks.	250	Focused heavily on a playtest, so more time was spent on making a solid prototype
Creation of Working Demo	80	~2 weeks to get a polished demo.	125	
Testing Working Demo	70	The goal is to make a smooth game.	125	

Table 6 - Game Design Person Hours

### Game Engine Hours

Task	Estimated Person Hours	Reason	Actual Person Hours	Reason
Product Research	70	~ 2 weeks to get initial research done.		
Engine Research	20	Want to do it right the first time.	20	

Math Research	30	Time needs to be spent on understanding the complexities.	60	Balancing between researching and implementing
Engine Selection	5	Each member has one one-hour meeting.	5	Simple Research
Math Selection	5	Each member has one one-hour meeting.	5	Trial and error mainly started with initial thoughts
Basic Rendering	80	Coupled with research = ~ 1 month time	115	General learning OpenGL (new technology)
Creating 2-D shapes	25	It is needed so that baseline information is there.	30	High effort throughout the team
Basic Ideas	30	Time to explore OpenGL and get used to the software.	35	Strong fundamentals were needed for the engine
Basic Math Concepts	25	The math is complex..	50	Lots of trial and error
Prototyping		Giving members ~1 week to complete each prototype.		
Sprites	10	1 week.	20	Much more involved for the prototype and iterating through prototypes
Klein Model	40	One month because, most important, this needs to be working exceptionally well.	20	
Camera Movement	10	1 week.	15	Relatively easy implementation
Lighting	10	1 week.	0	Unable to get to this point in the implementation
Assets	10	1 week.	12	General Sprites /Animation
Input	10	1 week.	16	Reimplementation required additional hours

Complex Issues		Because of their complexities, these tasks may take longer than a week to complete.		
Lighting	20	This may take more time to implement in a Non-Euclidean space.	0	Unable to get to this point in the implementation
Collision System	40	~ 2 weeks with two people.	0	Unable to get to this point in the implementation
Math Implementation	80	~ Everyone will spend 10-20 hours rendering complex math.	110	Lots of testing and redoing work due to complications
Demo Creation	200	Overestimate. Assuming previous tasks will take more time. Want to create a solid product. ~ 1 month and some.	120	Rescoped Demo accounted for fewer overall hours.
Creation of Working Demo	120		60	Many issues with implementation and performance. Rescoping and optimizing were required multiple times.
Testing Working Demo	80		60	

Table 7 - Render Engine Person Hours

### 3.7 OTHER RESOURCE REQUIREMENTS

In terms of game development, additional resources were needed, as discovered during the prototyping phase of the game design and development. The additional resources that were used were:

- Sprites, Images

For the game development itself, using sprites was required for the game development. Sprites allow for simple drawing operations to be performed over shapes rendered with the engine. Since none of the team were art students, acquiring sprites and images created by professionals would be preferable and result in a more polished and presentable game.

- Libraries for Engine Development

In developing the game engine, libraries must be used to ensure consistent code execution across platforms. Due to the nature of creating windows and processing input to write textures and shapes to the screen, various data structures must be used to copy data for the

Graphics Card. In addition to the general engine development required in the course's timeline, writing custom code to perform these tasks will be impossible. Therefore, using established libraries was incredibly important and helpful.

- Student Innovation Center Game Lab Access

During the game development prototyping process in Unity, members of the game design team have been unable to run prototypes on personal laptops efficiently. Accessing the Student Innovation Center Game Development lab would dramatically help develop, view, and create prototyping for Proof-of-Concepts.

- Unity Version Control

Unity Version Control (UVC) could be necessary for the Game Design team, as many game assets are huge files and will cause merge conflicts with other version control systems. The Game Design team has used the free version of UVC, which allows up to 5GB of storage space. However, funding may be required for the pro version of UVC as the final game may exceed the 5GB limit.

Overall, art, libraries, game lab access, and UVC were necessary resources for this project. Well-made sprites and images were needed to satisfy the aesthetic user requirements. Libraries were required to reduce redundant code being written. Student Innovation Center lab access assisted members of the game design team with prototyping on Unity. Finally, paying for UVC would increase our team's productivity and ensure the developers' smooth development.

# 4 Design

## 4.1 DESIGN CONTEXT

### 4.1.1 Broader Context

The design problem for this given project is in the context of computer graphics, game design, and mathematical modeling. The proposed solution seeks to innovate within the gaming industry by addressing the challenge of creating Non-Euclidean virtual worlds with accurate computational implementations. While this project primarily focuses on entertainment, it also has educational and research application implications.

The central communities this project is designed for are:

- Game Developers
- Gamers
- Academic and Research Communities

The primary audience for our project is both game developers and gamers. Game developers and those interested in pushing the boundaries of conventional game visuals are the primary focus for creating the game engine. The game is for gamers seeking novel and engaging experiences beyond traditional Euclidean constraints. Academic and research areas could be secondary audiences. The proposed game engine, which is primarily designed for games, could also be used by professors and mathematicians who want to visualize models in a Non-Euclidean space. It can also provide teachers a way to teach Non-Euclidean space with interactive media sources such as games or visuals, promoting the subject and gaining popularity.

The communities this project affects are:

- Gaming Industry
- Broader Gaming Community
- Academic Researchers

This project will affect the gaming industry because it can influence how game engines evolve to handle unconventional geometries, potentially inspiring a shift in the types of games being produced. It will also provide the industry with an engine to develop more games in this geometry, possibly increasing the number of users in this field and bringing it into the mainstream. The broader gaming community may also be affected. Players who experience the game might develop a deeper appreciation for abstract and unconventional spaces, which could foster interest in mathematical or scientific careers. Just as the secondary audience of mathematics may use this engine to visualize models, the project's API might be adapted for simulations in physics, mathematics, and other scientific domains.

The primary social needs that this project addresses:

- Innovation in Entertainment
- STEM Education
- Resource Optimization

- Accessibility of Advanced Concepts

This project aims to develop a genuinely Non-Euclidean game engine, offering players unique experiences and perspectives that redefine gaming possibilities. Additionally, users can gain fresh perspectives and continue to innovate the boundaries of what games can achieve in entertainment. Games utilizing Non-Euclidean geometry have the potential to become powerful educational tools, making complex spatial concepts easier to understand through immersive, interactive visualization. This project tackles broader challenges in computational resource management by focusing on efficient algorithms for rendering Non-Euclidean spaces, with potential applications in areas such as simulations and model visualization. Furthermore, introducing Non-Euclidean spaces to a mainstream audience helps demystify these advanced mathematical and scientific ideas, making them more accessible and engaging. This project's integration of technical innovation and creative engagement serves the gaming community and contributes to educational and computational advancements, making it impactful across several domains.

Area	Description	Examples
Public health, safety, and welfare	<p>This project primarily focuses on creating a novel gaming and educational tool, but has indirect implications for public health, safety, and welfare through its broader applications:</p> <ul style="list-style-type: none"> <li>• Mental Well-Being and Cognitive Development</li> <li>• Educational Impact</li> </ul>	By introducing players to engaging and thought-provoking Non-Euclidean worlds, the game can promote problem-solving, spatial reasoning, and creative thinking, which positively contribute to cognitive development and mental well-being.
Global, cultural, and social	<p>This project is designed to focus on inclusivity, ethical practices, and respect for the values of the diverse communities it affects. The critical aspects of how the project aligns with global, cultural, and social values:</p> <ul style="list-style-type: none"> <li>• Inclusivity and Accessibility</li> <li>• Respect for Community Practices</li> </ul>	<p>The game's emphasis on engaging and visually striking Non-Euclidean worlds ensures it can appeal to a global audience, transcending cultural and linguistic barriers.</p> <p>Implementing this project does not require or encourage changes to established community practices but instead introduces an optional, enriching tool for entertainment, learning, and creativity.</p>
Environmental	<p>This project aims to encourage players to plant and grow flowers and take care of the environment around them. Key aspects include the following:</p>	<p>Players gain positive connotations associated with growing crops and flowers by having users develop and maintain crops. The</p>

	<ul style="list-style-type: none"> <li>• Eco Friendly</li> <li>• Biodiversity</li> </ul>	<p>connotation may inspire users to grow plants and flowers and create gardens. Some potential benefits are increased biodiversity and providing bees with more pollen sources.</p>
Economic	<p>This project has various economic impacts at the individual, organizational, and broader community levels. Key aspects include the following:</p> <ul style="list-style-type: none"> <li>• Affordability for Consumers</li> <li>• Broader Market Impact</li> </ul>	<p>To ensure accessibility, the game and game engine must remain affordable for a broad audience, avoiding high price points that could alienate potential users.</p> <p>By pioneering a genuinely Non-Euclidean game engine, the project could establish a niche market, creating opportunities for partnerships, licensing, and expansion into educational and research sectors.</p>

*Table 8 - Broader Impact on Areas*

#### 4.1.2 Prior Work/Solutions

There are already some games that display in a Non-Euclidean manner. HyperRogue [4] is a Non-Euclidean roguelike game that uses exploration and combat as its primary gameplay focus. It uses the Minkowski hyperboloid model internally and the Poincaré disk model display.

Some of HyperRogue's Pros:

- Smooth gameplay
- Polished programming
- Non-Euclidean math

Some of HyperRogue's Cons:

- The bare bones of a game
- Poor UI and Design
- Crashes often
- Uninteresting gameplay

This proposed project plans to differentiate itself from this game by developing a solution with a solid UI that runs without crashing and has a fully fleshed-out game. This is planned to be accomplished by selecting a different genre of game. HyperRogue is a roguelike exploration game, whereas our proposed solution is a farming simulation game with exploration. The proposed genre increases player immersion, creating a better gaming experience.



Research was done to determine what models should be used to render the Non-Euclidean space. As used in HyperRogue, the Minkowski hyperboloid model and the Poincaré disk model display are used to visualize and store Non-Euclidean information. The Poincaré model is a mathematical representation or projection of hyperbolic geometry where points are located inside a unit circle, and "straight lines" are depicted as circular arcs intersecting the circle at right angles [6]. One feature of this model is that objects very close in the model might be very far away.



*Figure 7 - Example Poincaré Model*

The Minkowski hyperboloid is the underlying space hosting hyperbolic geometry, similar to how the surface of a sphere is the "true form" of spherical geometry [7]. This geometric space is often tough to visualize, requiring mathematical formulas for transformations between hyperbolic geometry and an Euclidean space that can be projected and rendered. This hyperboloid can be rotated with rotation matrices in the Minkowski space, like a sphere in Euclidean space with standard rotation matrices. This was why HyperRogue used it as its internal space for computations. The transformation between Poincaré and Minkowski is relatively straightforward. The hyperboloid can be rendered at the point  $(0,0,1)$  with the resulting projection being the Poincaré disk model.

#### 4.1.3 Technical Complexity

The proposed design is internally complex due to the game engine utilizing Non-Euclidean math. The general design is inherently complex due to the conversions between Non-Euclidean and Euclidean spaces. The complex calculations are primarily drawn from mathematical articles and papers, requiring a high mathematical understanding and many hours to read and understand the published articles and papers. Furthermore, the mathematical knowledge needed to understand many of these sources requires skills higher than calculus, primarily drawing from proof-based mathematical concepts. Another source of complexity is the new software used to develop the engine: C++, OpenGL, and various additional libraries. Many additional courses would be required to learn some languages, but the libraries are not used in any required or elective coursework. Therefore, much time must be allocated to learn the software and its intricacies. Additionally, the engine being developed must handle many different systems, such as but not limited to a resource management system to handle large amounts of memory requests and an entity component system to handle all the objects used in the game.

The game design itself also has its complexities. Like the game engine, development will occur in software unfamiliar to the group. Since the game will be run on the created engine, the internal components and interactions between different features in our game must be made from scratch. Some features that need to be implemented are object collision, lighting, pathfinding for monsters, player movement, and a system in which the player can farm. None of these features are trivial and needs to be implemented so that the game can run smoothly.

The proposed design is also externally complex. One of the requirements of our project is to handle different input types, such as keyboard or controller, due to the fact that this is the current standard for video game development. Additionally, current Non-Euclidean state-of-the-art games

are mainly for educational/research purposes. The plan is to exceed that by developing an enjoyable game. Another standard that needs to be met for the solution is runtime specifications. The game needs to be able to run at a reasonable framerate. Optimizations are a complex issue that cannot be solved trivially.

The proposed design has many interworking systems, each using mathematical and engineering principles to create a complex solution that renders a Non-Euclidean space. The design of our solution aims to match the industry standard in many areas, such as handling different input styles while exceeding other standards, such as enjoyable gameplay and well-designed gameplay relative to other Non-Euclidean games.

## 4.2 DESIGN EXPLORATION

### 4.2.1 Design Decisions

The primary design decisions discussed and finalized were related to the software used to develop the project and the general high-level design of the game that must be implemented. Regarding tooling, software was determined based on ease of learning and standard tooling relative to common engines used in the industry (regarding prototyping for the game designs). By choosing well-supported tools and developed software, many critical blocking factors were avoided, allowing us to work on the code instead of fixing the underlying tools used in the project. Additionally, mathematical models were evaluated similarly, primarily with the accessibility and relevance to the high-level game design. Choosing an appropriate model allowed the team to focus on a specific implementation, focusing our time and efforts instead of wasting time and effort throughout the semester. Therefore, we got basic shaders working and a few demos to prepare for deliverables. Finally, the high-level game design was determined based on team interest and providing an entertaining idea for the game design. Choosing an interesting game topic was by far the most critical decision. In game development, working on ideas people are passionate about is incredibly important, especially regarding large volumes of work. Since we were able to choose an interesting game idea, we continued to have high motivation throughout the year in the development process.

#### *Key Decisions*

##### **Software Usage (C++, OpenGL, and Unity)**

- Chose to implement the game engine in C++ with OpenGL graphics
  - Why?
    - A cross-platform language that can be built on multiple devices.
    - Good documentation, making it easy to learn with widespread library support.
    - Low learning curve relative to other graphics languages, allowing for fast learning.
    - State-of-the-art software.
    - Low-level resource management, which is beyond necessary for game development
  - Why is this Important?
    - Implementation will be in these languages, so choosing languages to support our requirements is imperative.
    - A good decision is needed to reduce learning times, and more time can be spent implementing features.
- Chose to prototype game features in Unity

- Why?
  - It is easy to learn, so less time is spent on learning and more on implementing.
  - Good resources allow for the use of internal sprites and objects for testing features instead of self-development.
  - Unity is a state-of-the-art system with many video games being developed using Unity.
- Why is this Important?
  - Low turnaround time allows for faster polishing of features, and more features can be developed.
  - Learning a state-of-the-art system is needed to know what must be implemented in the proposed engine.

### **Type of Math (Hyperboloid)**

- Chose to implement hyperbolic space
  - Why?
    - It is Non-Euclidean math, so it meets the basic technical requirements.
    - Has applications in the real-world field of cosmology.
    - It is a relatively well-researched mathematical field, so equations are openly available.
  - Why is this Important?
    - The project is, by definition, purely Non-Euclidean, so choosing a hyperbolic space is necessary.
    - Selecting a specific field narrows the scope with a limited time frame.

### **Game Genre (Farming/Exploration)**

- Choose to make a game in the farming/exploration genre
  - Why?
    - This would be the most enjoyable to implement based on the ideas generated for the main game.
    - It would be able to show off Non-Euclidean engines in an exciting way.
    - Experience playing games from this genre allows inspiration to be taken from well-established games.
    - Can implement features not seen in current games.
    - This genre typically has an interesting gameplay loop, a criterion of the project.
  - Why is this Important?
    - A genre is needed to create an enjoyable game (having too many genres makes the game feel too spread out, and not having a genre makes it feel out of place).
    - It makes further development and designing easier and allows for an anchor for further decisions.
    - Allows for more straightforward outside feedback.

### **Game Environment (Darkness/Horror-Lite)**

- Chose to design a game with a central theme of horror
  - Why?
    - A horror game allows for a different spin on farming simulators (most farming games are not horror).

- It promotes creativity in the design of characters and sprites.
- Horror aspects can evoke strong emotions from players, making the game memorable.
- Why is this Important?
  - Having a central theme ties the game together, making it seem cohesive.
  - It is another anchor for designing good NPCs, biomes, and other aspects of the game.
  - A good theme paves the path for how the user should feel while playing the game.

#### 4.2.2 Ideation

Product research was heavily relied upon to drive the decision-making process to decide what game genre should be implemented. The process was started by listing off games that the group determined as high quality due to the criteria of an enjoyable game. Focus was maintained on 2-dimensional games, as a technical decision already made was that a 2-dimensional game was to be made. For each game that was listed, reasons why these games were good and how these games could be improved upon were listed. Two main artifacts were derived from this list: a list of genres these games are from and essential criteria to meet while developing the game.

From this list, A brainstorming task was assigned where each team member created 2-3 ideas for pitching potential games. These ideas did not have to be fully developed, but what was important was the genre of the game, the central mechanic of the game, and the main theme of the game. All members did this to ensure all bases were covered. Using the information gathered from product research narrowed down what genres seemed better than others. After this brainstorming, five main ideas were fleshed out. These ideas were:

##### 1) Lights Out

A farming game that starts off peaceful and slowly becomes more sinister. The day is peaceful initially, but monsters can come out at night. Visibility at night is limited, but the player can place lanterns and other light sources to help them see. Specific light sources must be lit at night. Certain days/nights will have events. Otherwise, there will be tasks that the farmer should do at night, or they will lose crops/resources. Exploring further out at night will allow you to get more valuable resources.

Focus: farming, exploration, suspense/thriller, resource gathering, light

##### 2) Horrio

A 2D platformer exploration game. Generated dungeons that vary in theme with multiple layers/levels. Players can improve their stats and items as they progress. If you beat a dungeon, you can continue to the next, but if you die, you start at the beginning of the dungeon you are in. As you clear layers/levels in the dungeon, you get a roguelike reward (stats, items, etc).

Focus: Roguelike, Platforming, different themes, dungeon exploration

##### 3) Prison Break

You are a prisoner attempting to break out of prison. You can manage relationships, tunneling, smuggling, hiding from guards, gathering materials, craft tools, etc. Prison maps can be premade and generated with varying difficulty.

Focus: Puzzle, prison break, resource gathering/crafting

#### 4) Life Sentence

You're an aspiring crime lord attempting to rise to the top of an infamous crime syndicate. You will grow old as time passes, so you must hurry and gain power. If you get caught, you will lose years of your life in prison, bringing you closer to the end of your run. As you get convicted more, it becomes harder for your lawyers to reduce your sentence, resulting in more years lost. You will move drugs, take territory from rivals, and bank robberies, and choose when to make your move on those above you. The more scummy your tactics, the less respect your fellow criminals have for you. Watch out; when those you backstab get out of prison, they may come for you.

Focus: Speed, Criminal Activity, Reputation/Status

#### 5) Euclid-Shot

It is a fast-paced 2D bullet hell game where players navigate through intense waves of enemy projectiles, using their shots to cancel out hostile bullets in a strategic dance of survival. As players progress, they unlock unique classes and upgrades, each offering different playstyles and advantages, from high-speed evasion specialists to heavily fortified, slow-moving tanks. Roguelike elements mean each run feels fresh, with randomized upgrades and skill trees encouraging experimentation. In a Non-Euclidean space, projectiles will feel like they are flying from everywhere; bullets won't be traveling in a straight line.

Focus: Bullet hell, Roguelike, Gunplay, level-up

Another meeting was held to decide which option from these five options should be chosen. Detailed in section 4.2.3 is the process used to make the final decision. The option chosen was *Lights Out* due to its ability to utilize the Non-Euclidean space the best of the options provided.

### 4.2.3 Decision-Making and Trade-Off

A meeting was held to discuss the ideas above. The ideas were pitched to all group members to get new perspectives. From this meeting, any ideas needing more detail could be expanded. Also, additional clarification of each idea could be made in more detail. A weighted decision matrix decided which game idea would be chosen. Based on product research and the requirements from the project description, ten metrics were devised to measure the ideas. Each metric was given a weight between 1-5. As a team, each idea was ranked from 1-5, where five is the best and one is the worst of the ideas. Below is the matrix that was created.

Game Ideas												
	Non-Euclidean Utilization	Other Technical Requirements	Effort	User Needs Appeal	Exceeds State-of-the-art	Replayability	Fun Factor	Developer Appeal	Story	Originality		Final Weight
Weight	5.0	2.0	1.0	3.0	1.0	2.0	3.0	4.0	2.0	1.0		
Lights Out	4	4	3	4	3	4	5	5	5	4		103
Horrio	3	2	5	5	1	1	4	3	2	1		71
Prison Break	1	5	4	3	2	2	3	2	3	3		60
Life Sentence	2	1	2	2	4	5	2	4	4	5		69
Euclid-Shot	5	3	1	1	5	3	1	1	1	2		57

Figure 8 - Weighted Decision Matrix

Below is a description of each metric, along with the pros, cons, and tradeoffs of ideas for that metric and reasoning as to why the score was provided.

- Non-Euclidean Utilization

This metric is how well the idea would be able to show Non-Euclidean space. The score was higher if being in a Non-Euclidean space would allow an exciting feature to be added.

A weight of 5 was assigned since it is the leading technical requirement due to its relevance to the project.

*Euclid-Shot* received the highest score (five (5)) because a bullet hell where bullets travel in a Non-Euclidean manner makes the game unique and exciting. The light mechanic in *Lights Out*, seeing how lights would warp in a Non-Euclidean manner, allowed it to receive a four (4). The other games did not have a central feature that Non-Euclidean added to.

- Other Technical Requirements

This metric measured how effective each idea would be at showing off the other technical requirements of the proposed project. This was less emphasized because it did not refer to the main technical requirement.

The multitude of possible subsystems (e.g., relationships, entity A.I.) available in *Prison Break* would give it the best opportunity to show off the other requirements. The gameplay structure of *Life Sentence* made it challenging to find how other requirements could be met.

- Effort

The Effort metric quantified the estimated time needed to complete the game design of this idea. A lower score would indicate that a game would be too easy or difficult to make within the time frame.

Given the ability to do a micro-kernel development style for *Horrio*, it was determined to be the easiest to implement. *Euclid Shot* would be the most effort due to the number of projectiles rendered, meaning a heavier focus on resource optimization.

- User Needs Appeal

This metric measured how impactful the idea was at meeting the user needs defined in the early assignments of this class.

*Horrio* and *Lights Out* are in a genre of games that would meet the user needs defined earlier in this project. From product research, farming simulators, and platforming games were generally well-received by communities. Those games also allow for features to be added during development.

- Exceeds State-of-the-art

This metric evaluated how good the idea is compared to a state-of-the-art game in the same genre.

Given that Mario games heavily inspire *Horrio*, it was determined that creating a game that rivals Mario would be hard. *Euclid-Shot* would be the best possibility to make a great game

in the genre due to the Non-Euclidean space, giving it a significant difference from the other games in that genre.

- Replayability

This metric quantified how much this game idea could be replayed. From product research, games with a replayable factor are generally more enjoyable.

In the current state of video games, a game like *Horrio* would be played once and never again. The unique gameplay experience of *Life Sentence* and its reincarnation feature would provide the best replayability experience.

- Fun Factor

This metric is how fun a game would be given the idea.

Even though *Euclid-Shot* would be cool to develop, it may not be fun due to the limiting factor of rendering optimizations. Also, that type of genre is only enjoyed by a small community. *Lights Out*'s unique theme and gameplay loop ideas make for a fun experience accessible to many users.

- Developer Appeal

This metric evaluated how invested, as developers, this group was toward the idea.

*Euclid-Shot* was initially attractive, but after careful consideration about what would be developed, interest was lost. *Prison Break* and *Horrio* are solid ideas, but would be too similar to other games. This group is invested in *Lights Out* and how a horror theme draws on users' emotions.

- Story

This metric measures how suitable a story can be told in the idea.

*Lights Out* has the most ability out of the ideas to tell a compelling story. The genre leads to a game being driven by story, and a shadowy environment allows for interesting characters to appear. Games like *Horrio* and *Euclid-Shot* would have little story and, thus, a low score.

- Originality

This metric measured how original this idea is compared to other games.

The games heavily influenced by other games, such as *Horrio*, were rated lower. *Life Sentence* had the least influence from another game, so it got the highest score.

From this matrix, *Lights Out* is the game best suited for continued development. It scored highly on all metrics, so it was the best option. Given the style of the game, other features from other ideas can be easily incorporated. The importance of making a fun, enjoyable experience, this game allows for the best opportunity to do that.

## 4.3 PROPOSED DESIGN

### 4.3.1 Overview

#### *Game Design - Lights Out*

A farming game that starts off peaceful and slowly becomes more sinister. The day is peaceful initially, but monsters can come out at night. Visibility at night is minimal, but the player can place lanterns and other light sources to help them see. Specific light sources must be lit at night. Certain days/nights will have events; otherwise, there will be tasks that the farmer should do at night, or they will lose crops/resources. Exploring further out at night will allow you to get more valuable resources.

The main plot of the game:

You received a letter from your grandpa one day telling you to meet him at his farm because he found something exciting he wanted to share with you. Knowing his farm is far from any civilization and in the middle of a forest, you pack a bag and take off. As you get to his farm, something seems off. This sort of black fog is off in the distance in the forest. As you knock on the door, there is no response. Suddenly, a monster from the forest attacks you. As you flee, a stranger traps the monster. They explain to you that your grandpa is missing and you must find them. It is up to you and your basic farming knowledge to grow resources to find your grandpa in a not-so-normal forest. You must be quick, as the monsters keep coming after you.

Main Mechanics of the Game:

- Farming

The main mechanic is farming plants to create resources to use. However, the plants you grow are not your typical plants. Some plants create light, some create shadows, and others create rocks. Different seeds are found throughout the world, and only in certain areas. Plant growth is based on watering and timing.

- Exploration

Another main mechanic is exploring a Non-Euclidean space. No map will be given, so users must learn how to traverse it themselves. Different biomes will have unique plants, enemies, resources, and characters. As players explore deeper and deeper, they will get better resources and face more challenging monsters.

- Enemies

There are monsters dispersed throughout the area. They speak their own language, which the player does not understand. Each enemy has its own style. The monster's primary goal will be to attack the player.

- Traps

To prevent enemies from killing you, you must trap them. There is no standard combat system in this game. You must grow plants to create traps to stop them. Once you have created one trap, you can use it indefinitely.



- Light

The world has a day/night cycle. At night, everything is pitch black unless a light source is placed. These light sources are used to ward off monsters at your farm. Before each night, you must light each lamp; otherwise, monsters might destroy what you have created.

Different light sources give off differing amounts of light.

### *Game Engine*

The game engine supports the requirements specified by the game design team. Individual submodules supported operations by managing memory, handling inputs, and drawing to the screen. The engine will generally operate on a loop, handling inputs from the screen and outputting button presses and shapes to the screen.

The engine was built to serve as a scalable system that can support future development and shifting design requirements. Using CMake, building the system is trivial, with source code files designated in a centralized location for compilation. With the intended implementation, adding new features is streamlined and straightforward.

Furthermore, external libraries are supported and encouraged to be used due to the ease of integration and implementation that is possible using the libraries. Using external third-party libraries, creating core functionality from scratch is not required or expected.

The core design principle we're using is the Entity-Component-System (ECS) as the backing manager for resources. ENT is the specific ECS that was integrated due to how well established it is and the heavy optimization it has previously undergone. The ECS is a necessary component due to the many different subsystems used throughout the engine. Since a single entity has many other data fields operated on by the various subsystems, the ECS remediates issues inherent to Object-Oriented languages, where the entire object (and all the unnecessary components) are sent to each subsystem.

With the ECS, a system scheduler is used to sort and prioritize the execution of the subsystems for optimal operation throughout the core processing loop. This is done through a specific data structure that maps dependencies for each subsystem and executes updates in the most optimal order. The ECS hosts all the core systems for operation and allows the scheduler to access and optimize the order of updates for the system as a whole.

The last core component is the Rendering Subsystem and the Non-Euclidean Shaders. The project's purpose was to create our own custom shaders and to support the shaders through the engine. The Rendering subsystem allows for the variable use of different shader pipelines to be displayed to the user. Therefore, many different shaders can be used as long as they are included in the project, allowing for much more freedom than traditional game engines (e.g., Unity).

### 4.3.2 Detailed Design and Visual(s)

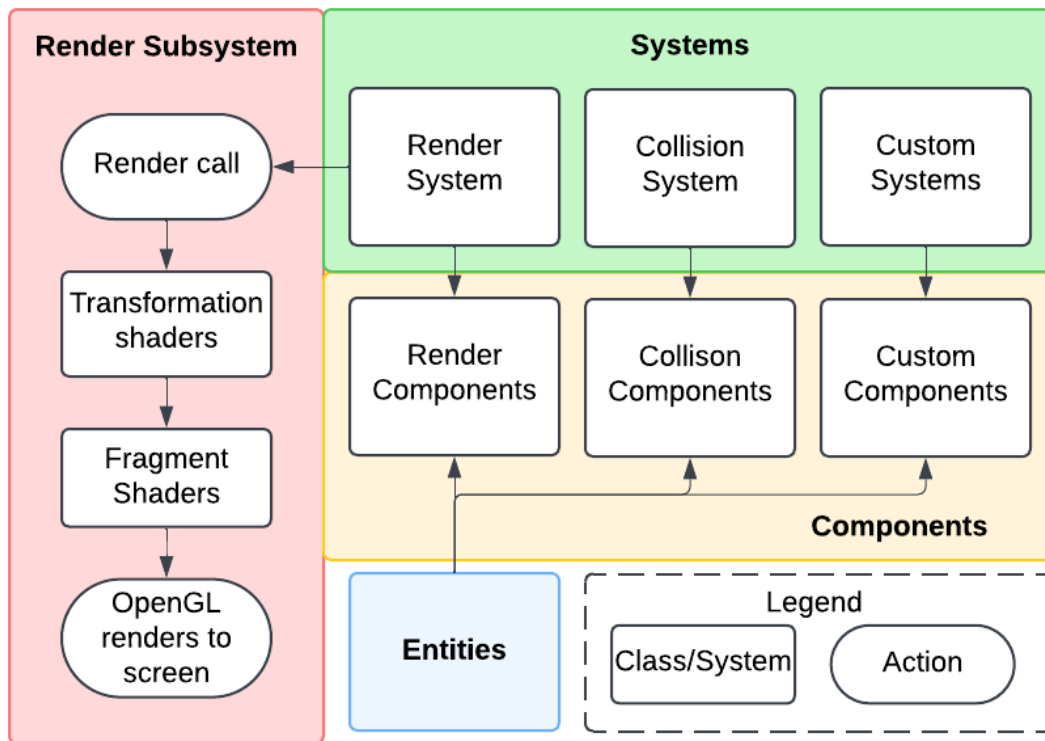


Figure 9 - Detailed Design and Visuals

#### The Game Engine

The game engine will process inputs from the game to project and respond to actions performed in the game effectively. Shaders will perform calculations to convert the Non-Euclidean world to Euclidean coordinates to be rendered to the screen. The Entity-Component-System will manage resources while the primary rendering loop will draw objects to the screen with helper classes to handle inputs, configure settings, perform actions, and facilitate interactions between objects in the world. Furthermore, the engine will support a top-down 2-dimensional projection of the world.

#### Entity Component System

The Entity-Component-System (ECS) maintains the memory allocation of all the required sprites and entities. Some user inputs will require the system to load objects from memory to be used. Computer graphics have easily-predicted memory accesses, so having a system in charge will allow for better performance and remove the overhead associated with default C++ memory allocation (**malloc**). Furthermore, objects in game development cannot be designed similarly to normal object-oriented programming due to the many data fields being used by various subsystems. Therefore, the ECS will manage the data fields while they are in use by the subsystems.

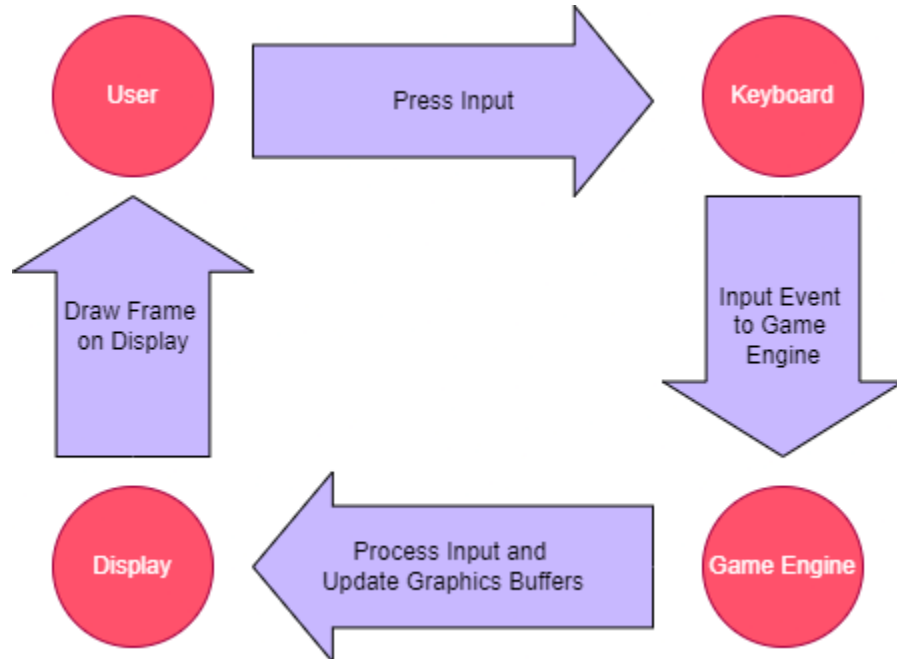
#### Laptop

The laptop is the primary I/O device for the user and the system on which the Game Engine, ECS, and Game Engine will run. A laptop will consist of a keyboard that will be used as the input. It will

send these inputs to the game engine. It will receive data from the engine, a new frame that the laptop can display on its screen, so that the user can see. The laptop itself is a specific design choice due to the generally lower power applications, including limited computational resources. The limitations intentionally force optimizations to increase the number of playable platforms, leading to increased accessibility.

### 4.3.3 Functionality

#### *System Loop*



*Figure 10 - User-System Feedback Loop*

The user will interact with the proposed solution using a computer and some input device (keyboard or controller). Players interact with elements such as moving the character, selecting inputs, or triggering actions when playing the game. These actions will be rendered in real time by the engine. As the player navigates the game world, the engine continuously processes data from the game logic (e.g., the character's position, environmental changes, interactions with objects) to produce and render frames to the display.

The game engine operates as the visual backbone of the game, responding to the player's actions to deliver an immersive experience. Below are examples of what the game engine should be able to do:

1. **Scene Initialization:** The player loads into a dimly lit forest scene. The render engine loads assets, applies local lighting, and renders the world.
2. **User Interaction:** The player moves toward a light seen in a cave. The lighting gradually increases as the player gets closer to the entrance.
3. **Scene Transition:** The player enters the cave and creates a new environment. The engine adjusts brightness, assets, sprites, and detail levels for the new environment.

These visual frames will be the primary output for the system. Based on the updated frames, the user can make decisions in real-time and continue this feedback loop as long as the user wants to.

#### 4.3.4 Areas of Concern and Development

The current design satisfies the requirements and meets the user's needs. Additional features will need to be implemented to continue meeting the needs, but based on the feedback received from user testing, these requirements were met. Users' needs and technical requirements are being continuously updated based on what has been created.

The current primary concerns for developing this product are:

- Meeting user design constraints when it comes to artistic choices.
- Creating functional and entertaining game mechanics surrounding the farming aspects of the game.
- Time constraints and integration between the game design and the engine.
- Performance issues due to inefficient processing methodology.

The plan used to develop solutions to address those concerns was to meet with our advisor to get guidance on how valid these concerns are. Scope readjustment was discussed to determine the feasibility of a project that seems completable within the time constraints. Group meetings were scheduled to determine if any changes to the design are needed. The central questions that were asked to advisors were clarifying if they believed, given our progress, that the constraints can be met, and asking internally if the design has met the user needs. Some extra concerns this team had while implementing our design were the feasibility of creating a working demo within the timeframe. Due to delays detailed earlier in this document, there was less time than initially planned to create a polished demo.

#### 4.4 TECHNOLOGY CONSIDERATIONS

Technology	Pros	Cons
OpenGL	State of the Art Easy Rendering Language to Learn Cross Platform	Performance variability between platforms
Unity	State of the Art “Free” to use Generic use Documentation/Resources	Not customizable at the engine level Poor optimization for what is trying to be accomplished
GitHub/Unity Version Control (UVC)	Version Control Task Allocation/Management	Github–limited file size Github–does not handle Unity resources very well UVC–not free
Personal Laptop	Ease of Use Easy to Test Easy to transport	Limited computing resources Take off (in class)

*Table 9 - Technology Considerations*

Above are tradeoffs that were made in determining the technology being used. The decision was made to use OpenGL as the graphics language, which allows cross-platform rendering – a vital factor to increase accessibility by hosting multiple platforms. However, performance degradation

may occur due to platform-specific issues. An alternative choice could have been to develop in another language that only works on one platform, with better performance. Still, the cross-platform adaptability is more essential to meet the requirements. Another choice was to prototype the game design in Unity, allowing for more progress before the game engine was complete. The trade-off is having to spend time integrating the code written in Unity to meet the interface of the render engine. One alternative would have been to create the render engine entirely and then build the game on that. This was decided against due to the timing constant. Finally, another choice was that the solution needs to run on a personal laptop. This allows for better ease of use. However, the trade-off is that the game will have to run on some limitations of resources. A different solution would be to make this game only run on computers with high-quality hardware parts. Still, the downside to this solution is that it may be harder to test those specs and decreased accessibility due to financial limitations.

## 5 Testing

In the current state of this project, multiple different systems are to be tested. In terms of game design, this section will detail how this group implemented unit tests for each mechanic of the game that has been designed. It will detail how these mechanics interact with each other and how interface testing will be used to validate those interactions. The integration testing tests how the game functions in a single scene with all the mechanics regarding the render engine. An important section for testing the game was user testing, in which this group conducted a playtest, in which a polished version of the game was built and sent off to prospective users to receive feedback on whether or not the state of development met the users' needs and requirements.

The engine systems components for testing focused heavily on using existing frameworks like Catch2 and CTest for unit testing, as well as other levels of testing like Interface and Integration. Regression testing ensured that when additional subcomponents were implemented, they did not break previously implemented components. Due to the lack of time, there was limited user testing of the engine, and future iterations of this project would focus on user and acceptance testing.

### 5.1 UNIT TESTING

Each unit necessary for gameplay must be tested throughout development and before initial release builds. Regarding the specific units to be tested, the currently developed units are Camera, Input, InputManager, and Entity-Component-System classes. All of the Game Engine must be thoroughly tested before gameplay development. Additionally, each component of the actual game must be tested, which will inevitably build off the game engine itself. Therefore, thorough testing of the Game Engine was required.

For the prototypes in Unity, the Unity Testing Framework (UTF) is an incredibly robust testing suite that allows for simple unit tests in Unity and general manual tests for correctness. Another critical portion of the game itself was how the gameplay feels, which is very subjective. Therefore, manual testing was required for polishing gameplay and gathering opinions on the gameplay loop and mechanics. The edit mode of UTF was used to ensure that objects and assets have the correct properties and settings. The play mode was crucial for validating the correctness of functionality.

Manual tests will also give basic gameplay mechanics. Each mechanic implemented was created separately in Unity in its own space. It was vetted that the mechanic worked separately through manual testing and playing through to make sure that it met the requirements of the mechanic. This ensured that before integrating mechanics together, each mechanic worked as intended.

The Engine was tested using Catch2 and CTest for unit tests. The game engine was much more straightforward to test due to its functionality and demanding performance requirements. Correctness is critical, so using vetted testing suites for C++ to ensure correct values are output for each functional unit is ideal. CTest and Catch2 are very popular tools for C++ project testing. These tools were also chosen due to their functionality and integration capabilities.

### 5.2 INTERFACE TESTING

The interfaces in the design primarily involved the Game Engine and the Game Design portions of the project. The game does not connect to a server, meaning many interfaces will involve components internal to the project. For example, the primary gameplay loop has to interface with Input Managers and Shaders, and the correctness of communication will have to be ensured through these tests.

Regarding the finalized product, the primary testing methods will be playtesting and CTest with Catch2 for the unit tests and interface between components. The highly interconnected nature of the gameplay and the classes requires close integration and efficient communication between objects and methods.

## 5.3 INTEGRATION TESTING

The critical integration paths within the design were in the game engine, with response time in latency to ensure smooth gameplay. Due to the correctness ensured by unit tests, integration tests ensure that the individual components efficiently work together to meet performance requirements for a smooth gameplay experience. This testing section is of utmost importance to the success of this project.

Integration tests will be imperative in developing the final deliverable, from porting over Unity code to the custom game engine. Once the initial unit tests are completed, stringing together individual parts must be tested to ensure new additions are working as intended. Furthermore, monitoring performance will be critical to identify early in the development lifecycle.

Catch2 can once again be used in integration testing due to the robust nature of the suite, along with the fact that the scalability of tests can be extended from the interface tests to the integration tests. An issue to focus on in integration tests is ensuring that the results of interface tests are not drastically changed once integrations occur, meaning that the high-level focus areas concentrate on outputs all being correct once integrated together, preventing overlapping effects from method calls.

## 5.4 SYSTEM TESTING

System-level testing will primarily involve performance requirements and gameplay mechanics as the primary focus of the system testing. Once the integration tests are completed for correctness, performance targets will be met, using profilers to observe targets for optimizations. Additionally, playtesting will have to occur to determine if mechanics need to improve (i.e., whether certain mechanics need to be smoother or feel good). One of the most important requirements is to create an enjoyable game, and the mechanics, design, and performance are the most important factors from a user standpoint for an enjoyable game.

Integration-level testing can also be run in parallel to identify bugs or internal logic issues during gameplay. During system testing, the integration tests are expected to pass correctly. Still, there are many cases in longer playtests or sessions that can lead to issues with memory leaks or general performance issues. Profiling can identify sources of memory leaks and analyze the primary compute-heavy areas. Additionally, Nvidia Insight can also perform similar functionality for shader performance. Still, given that the game is 2-dimensional, the likelihood of graphics card bottlenecks was very low, even with many mathematical conversions.

## 5.5 REGRESSION TESTING

Regression testing occurred similarly to the integration tests due to the very similar nature of the testing phases. Once changes are made, the existing functionality must be verified to ensure correctness and meet performance and entertainment requirements. Additionally, playtesting can

filter bugs similar to alpha and beta builds in many mainstream games. Furthermore, outreach to hear many opinions on the gameplay to ensure our base requirement of having an enjoyable and unique experience will be essential for this testing phase.

Another vital piece to consider is performance requirements within the new updates. Profiling will continue to be essential to observe performance gains or losses for fixes to occur and be planned. Many optimizations will be required in the later stages once the correctness of gameplay is tested and vetted, further emphasizing the need for profiling.

## 5.6 ACCEPTANCE TESTING

Acceptance testing will occur after all previous tests have been completed. Therefore, all functional and non-functional requirements will be met at this time. However, code reviews happened at this stage, with proper documentation required. Furthermore, each performance target was re-evaluated at this stage, primarily meeting high-level framerate targets and memory requirements.

Written code examined at this point was required to meet style guidelines as discussed by the team, with clean and well-written code subject to reviewer discretion. After all issues are met, the code will be pushed into the main branch. Additionally, beta tests of the mechanics and features added will be performed at this stage to ensure polished animations and responsive actions are present in the update.

## 5.7 USER TESTING

To determine whether or not our design meets user needs, this group designed a playtest of the game. A playtest is a critical stage in the game development process where a prototype or near-finished version of the game is made available to players outside the development team to gather feedback and identify areas for improvement. To test whether our design addressed user needs, we provided players with clear instructions, intuitive controls, and a goal-driven narrative. Then, we observed how they interacted with the game in real time. Users were not given a complete game but a portion of it to make the playtest short and digestible. Keeping it limited in scope allowed the group to get better results. During these sessions, we observed how easily players navigated the farm and forest environments, how they understood the mechanics like planting, exploring, and using traps or lanterns.

From the playtest feedback, we identified three consistent issues: the first being frustration with the loss of plants when dying/exploring the forest, then unclear interactions, and accidental inputs. These insights helped us spot moments when the players' expectations clashed with the current mechanics. In response, we have plans to reinforce the permanence of planted crops to reduce player frustration, include visual indicators to clarify interactions with the market, and adjust the control mappings to separate overlapping inputs. These changes aim to preserve the intended design while reducing some of the frustrations in the current player experience.

## 5.8 RESULTS

Throughout development, this team employed a comprehensive testing strategy encompassing unit, interface, integration, system, regression, and user testing to ensure the engine and gameplay met functional, performance, and user experience requirements. Unit testing was conducted using



Catch2 and CTest for core engine components like Camera, Input, and ECS systems. In contrast, Unity's Testing Framework (UTF) validated individual gameplay mechanics and asset configurations. Interface testing verified correct communication between internal systems, such as input handling and rendering. Integration testing ensured all components worked smoothly together, especially during the transition from Unity prototypes to the custom engine, with a focus on maintaining performance and responsiveness. System testing focused on overall performance and gameplay feel, using profilers to detect bottlenecks and memory issues. Regression testing ensured that new features didn't break existing functionality, and acceptance testing confirmed that final builds met all functional, performance, and code quality standards before merging into the main branch. Finally, user testing through structured playtests validated whether the game met user expectations and provided an enjoyable experience, guiding final design refinements.

The results from our comprehensive testing process revealed several key insights that guided the refinement of both the game engine and gameplay experience. Unit tests confirmed the stability and correctness of core systems like input handling, camera movement, and entity management. In contrast, interface and integration tests validated smooth communication and performance across systems, especially during transitions between scenes and mechanic interactions. System testing identified minor performance issues, particularly related to memory usage and responsiveness, which were addressed through profiling and optimization. Regression testing ensured that newly added features did not introduce bugs or break existing functionality. User testing, through focused playtests, revealed that players generally found the controls intuitive and the core gameplay loop engaging. However, some mechanics—such as seed switching and enemy avoidance—required further clarification or balancing. Overall, the testing process confirmed that the project was on track to meet its goals, highlighting areas for continued improvement and polish.

Future progress can be made in this area once more progress has been made in integration. User testing was limited for the engine due to a lack of a working product for much of the process. Now that something tangible exists to work with, more extensive user and acceptance testing can be done on the system. It may also be warranted to attempt different projects on this system now, just as this team showed off the engine.

# 6 Implementation

Implementations are divided by section, with engine implementations separate from the game design implementations. Game design prototypes have been implemented in Unity to determine initial mechanics and gameplay.

## 6.1 GAME DESIGN IMPLEMENTATIONS

The game design implementation process is broken down into a series of stages:

- Prototyping of basic functionality.
- Integration of prototypes with each other.
- Development of scenes.

Breaking the implementation stages down into these allows for check-ins to happen at the end of a stage to determine the project's feasibility and if the timeline needs to be readjusted. These stages also correlate with the different levels of testing that will occur. Prototyping will mainly relate to unit testing, whereas integration and development of the scene will be part of integration and system testing, respectively. The current state of progress for game design is between the second and third stages, which entails fine-tuning the individual prototypes to work together better in the scene.

### Prototyping

This first stage focuses on creating prototypes for the essential mechanics of the game. The mechanics focused on here were the player, tile mapping, monsters, lighting, and movement. A very base-level working demonstration can be achieved if these mechanics can be implemented. These prototypes were developed in Unity, and each member of the game design team was initially assigned one of these mechanics to prototype. There was a two-week timeline to get this done. After this time, each member presented their demo to the rest of the team to get feedback and allow members to see what the other were doing. If the prototype met the project's requirements, members would go on to either the next mechanic prototype or being on the next stage.

### Integration

Once enough main mechanics were prototyped, the second implementation stage could begin. This stage focused on the integration of mechanics and their interactions. Conceptually, this is very straightforward and combines the different parts and modifies certain aspects to work together seamlessly. The vital part of this stage was ensuring that when integration was happening, sprites and entities were consistent and that there were not two different ways of tiling, for example. This stage also focuses on unifying the team's code standards.

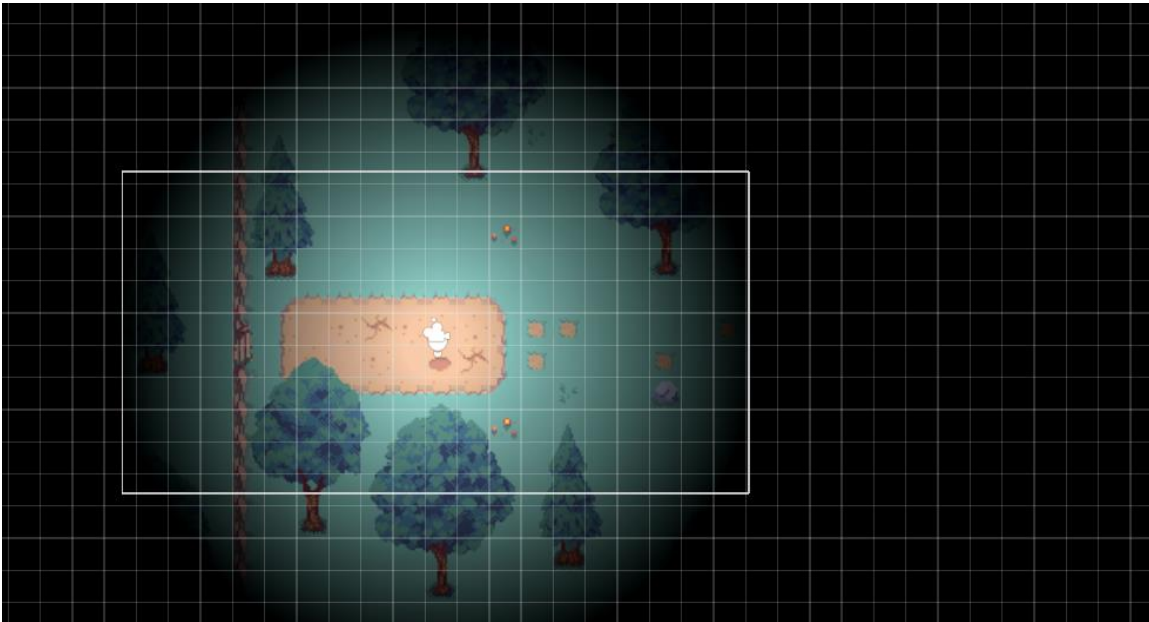
### Scene Development

In the third stage, work was done to create scenes. Two scenes were created for the faculty presentation. These scenes are the home farm scene and a forest scene. All of the mechanics that have been implemented are used in these scenes.



*Figure 11 - Home Scene*

The home scene is where the player will start the game. It consists of a house, a silo, and several plots of land. The house serves as just a simple image, with no deeper functionality now. The silo will be made to be interactive. This will let the user store items there to keep things organized since the player has limited immediate inventory space. The plots of land will serve as interactive fields, in which the player can grow plants for future use. This overall scene can be left for the next by taking the stairs seen to the far right. These straits will cause the player to transition to the next scene.



*Figure 12 - Forest Scene*

The forest scene, picture above, introduces the idea that the player needs some light source. Currently, it is implemented such that upon entering, the player is plunged into a forest so dark that he can only see in a small bubble around him. To get through, the player will need to place lanterns and explore. While exploring, the player must watch out for monsters that will avoid lanterns, but not the immediate field of vision displayed currently.

### Current Status

The current state of game development consists of a game with multiple areas, enriched with Non-Playable Characters (NPCs), fully implemented mechanics, each has undergone rounds of testing to improve features. It is not a completed game, as time limited how much could be implemented. Future work in the game development area includes adding the blue forest area, the slip n' slide area, and fleshing out the story/ lore.

The full list of completed functionality is as follows:

- Game Ideation
- Mechanics
  - Farming
  - Lighting
  - Monsters
  - Inventory Management
  - Sounds
  - Shops
  - Traps
  - Seeds Types
- Home Scene
- Forest Scene

- Player Movement
- Prototype Demo
- Story development
- Lore creation

## Monsters



*Figure 13 - Forest Scene with Monsters*

While exploring the forest, the player will encounter some monsters. These monsters will chase the player throughout the forest. If a monster comes into contact with a player, the player will die. A player may place a trap, and when a monster runs onto a trap will be locked in place and can no longer chase the player. Monsters use A\* pathfinding algorithm to find players. Players explore the forest for rare seeds that can be grown into unique plants, traded in the market for items, or used to progress towards the end of the game.

## Shops

Shops are a part of the game used to convert the seeds you gain from farming into valuable materials. Users can interact with the shops, and if they have enough seeds in their inventory, they will conduct a trade; if they don't, a UI text box will appear telling them they need to collect some more seeds. These are made with a shop management system class that, when the user presses the correct button within a specific distance of the shop, checks a condition if the current player's inventory possesses the right amount and type of seed, a transaction event occurs where the player loses those seeds but gains the item.



## Your a few seeds short, bring

*Figure 14 - Shopping Area in Game*

A list of the functionality that has been implemented in the engine for the game

- Player Movements
- Sprites
- Forest Scene
- Monster Mechanic

There are things that this team was unable to accomplish in this time frame. That includes making the blue forest scene, the slip n slide scene, and implementing more traps, seeds, and monsters. This team was also unable to finish the story and lore of the game. This team also only had time to implement the forest scene and the mechanics relating to the forest. The rest of the scenes and mechanics must also be implemented in the engine.

The main reason this team was unable to accomplish these tasks was that this team ran out of time. Specific tasks like user testing and demo creation took longer than expected, as well as realizing that the initial scope of this project was more ambitious than what could have been accomplished in this timeframe. It is reasonable to assume that if given more time to perform these tasks, they all would be accomplished due to no other blockers.

## 6.2 ENGINE IMPLEMENTATIONS

### Current Status

The game engine is currently in an integration-ready state with the core functionality present, but it needs polishing. Each of the main subsystems allows for a working Euclidean prototype, with heavy optimization required for the tile map rendering portion of the workflow due to lacking techniques in the Non-Euclidean Shaders and determining which tiles to render in the map itself.

The full list of completed functionality is as follows:

- Entity-Component-System (ECS) Integration – ENTT library
- Input Management
- Resource Management
- Sprite Sheet Loading and Display
- System Scheduling
- Custom Component Compatibility
- Offline Storage
- Euclidean, 2-Dimensional Display (GUI and text displaying)
- Custom Developer Tooling:
  - Texture Atlasing
  - Key Mapping Management
  - Animation Editor
- Custom Tile Mapping (Non-Euclidean and Euclidean Compatible)
- Render Loop Abstraction
- Non-Euclidean Shader



*Figure 15 - Forest Scene in Engine*

A current image of the game being rendered on the engine itself is seen above. The design currently renders a {4,5} tessellation mapping with a layer depth of 8. This equates to 3051 tiles being rendered on the screen at once. This team also has z-layer depth implemented, meaning that the rocks and the player seen on the screen are different entities that are rendered above the tiles. This means that more than 3000 things are being rendered at any given frame. To improve performance, our system scheduler will batch tiles that are similar and close to each other, reducing the amount of time our buffer gets allocated and the number of draws. The tiles are generated using a BFS based on the center tile.

Although most of the core functionality was completed, we're still missing a few core functions from the engine, which is acceptable due to the mismatching timeline with the game design team. If we were to continue the project itself, these would eventually be completed, but they aren't necessary at this point. These functions are:



- Sound Support
- Collision Detection
- Different Tessellation Support
- Improving Map Generation

These were not completed due to the project's timeline and other features taking priority, especially when considering the timeline of the Game Design Team. Due to being unable to start integration with the Unity demo, the team did not finish these features, focusing on other features necessary for the demo itself. Despite this, a barebones, working engine was completed with enough functionality to support the required demo to be used for the final deliverables.

## Math

This section will mention the math, including projecting the Poincaré model we used onto hyperbolic space and how we transformed the hyperbolic space onto a Poincaré model. This section will also include the rotation matrix used to rotate the tiles around the hyperbolic space.

Poincare (2D)  $\rightarrow$  Hyperboloid (3D)

$$\begin{pmatrix} x_h \\ y_h \\ z_h \end{pmatrix} = \frac{1}{1 - x_p^2 - y_p^2} \begin{pmatrix} 2x_p \\ 2y_p \\ 1 + x_p^2 + y_p^2 \end{pmatrix}$$

Figure 16 - Equation from Poincaré to Hyperboloid

Hyperboloid (3D)  $\rightarrow$  Poincare

$$\begin{pmatrix} x_p \\ y_p \end{pmatrix} = \frac{1}{1 + z_h} \begin{pmatrix} x_h \\ y_h \end{pmatrix}$$

Figure 17 - Equation from Hyperboloid to Poincaré

The Poincaré disk and hyperboloid models are two equivalent representations of 2D hyperbolic geometry, related by a smooth, invertible mapping. The Poincaré model embeds the hyperbolic plane inside the unit disk, where geodesics appear as circular arcs orthogonal to the boundary. The hyperboloid model, on the other hand, represents the hyperbolic plane as a two-sheeted hyperboloid in 3D Minkowski space (with coordinates (x,y,z)) defined by the equation above.

This transformation arises from projecting the hyperboloid onto the unit disk via stereographic projection from the point (0,0,-1) onto the plane z=0. The inverse map returns Poincaré coordinates

to the hyperboloid, preserving hyperbolic distances. This correspondence is crucial in physics and geometry, as the hyperboloid model simplifies certain computations, especially involving isometries and Lorentz transformations.

Hyperbolic Rotation about x ( $R_x$ )

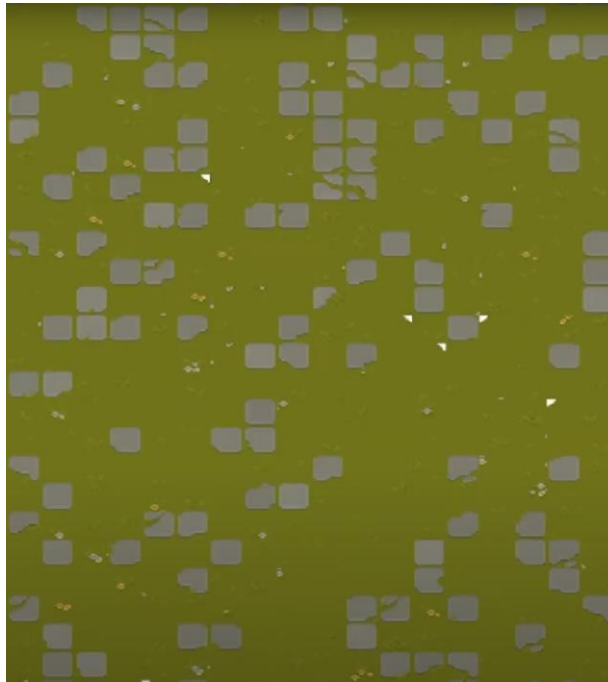
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cosh(\theta) & \sinh(\theta) \\ 0 & \sinh(\theta) & \cosh(\theta) \end{bmatrix}$$

*Figure 18 - Equation detailing Hyperbolic Rotation about x*

The matrix above represents a hyperbolic translation along the y-axis in the hyperboloid model of 2D hyperbolic geometry. It ensures that points on the hyperboloid remain on the hyperboloid after transformation. The top-left value of 1 indicates that the x-axis remains fixed. At the same time, the lower 2x2 block forms a hyperbolic rotation in the y-z plane, characterized by hyperbolic cosine and sine functions. The parameter theta determines the magnitude of translation in hyperbolic distance. This matrix is a core element of the isometry group, and geometrically, it moves points along a geodesic through the origin without rotating them. Such transformations are fundamental in generating hyperbolic tiling's and modeling motion in negatively curved spaces.

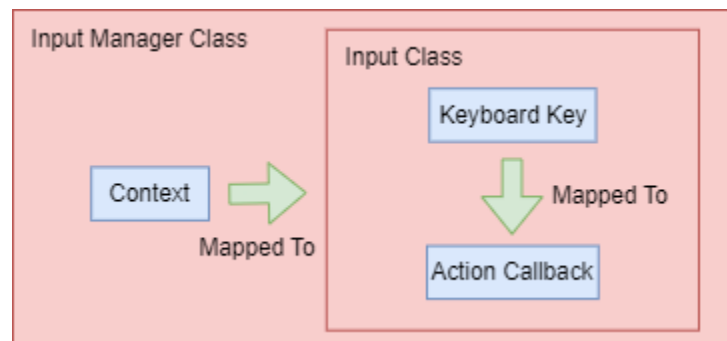
### Example Prototypes

Detailed below are a couple of prototypes that were developed over the course of this project. They show how this team went about developing certain subcomponents for the system.



*Figure 19 - Sprite Sheet Render Tiles Example*

Additionally, sprite rendering and entity-component-system prototypes are in progress, with major optimizations required to be added. Figure 11 demonstrates example sprites rendering as tiles to serve as a base for the world.



*Figure 20 - Input Management Mapping Scheme*

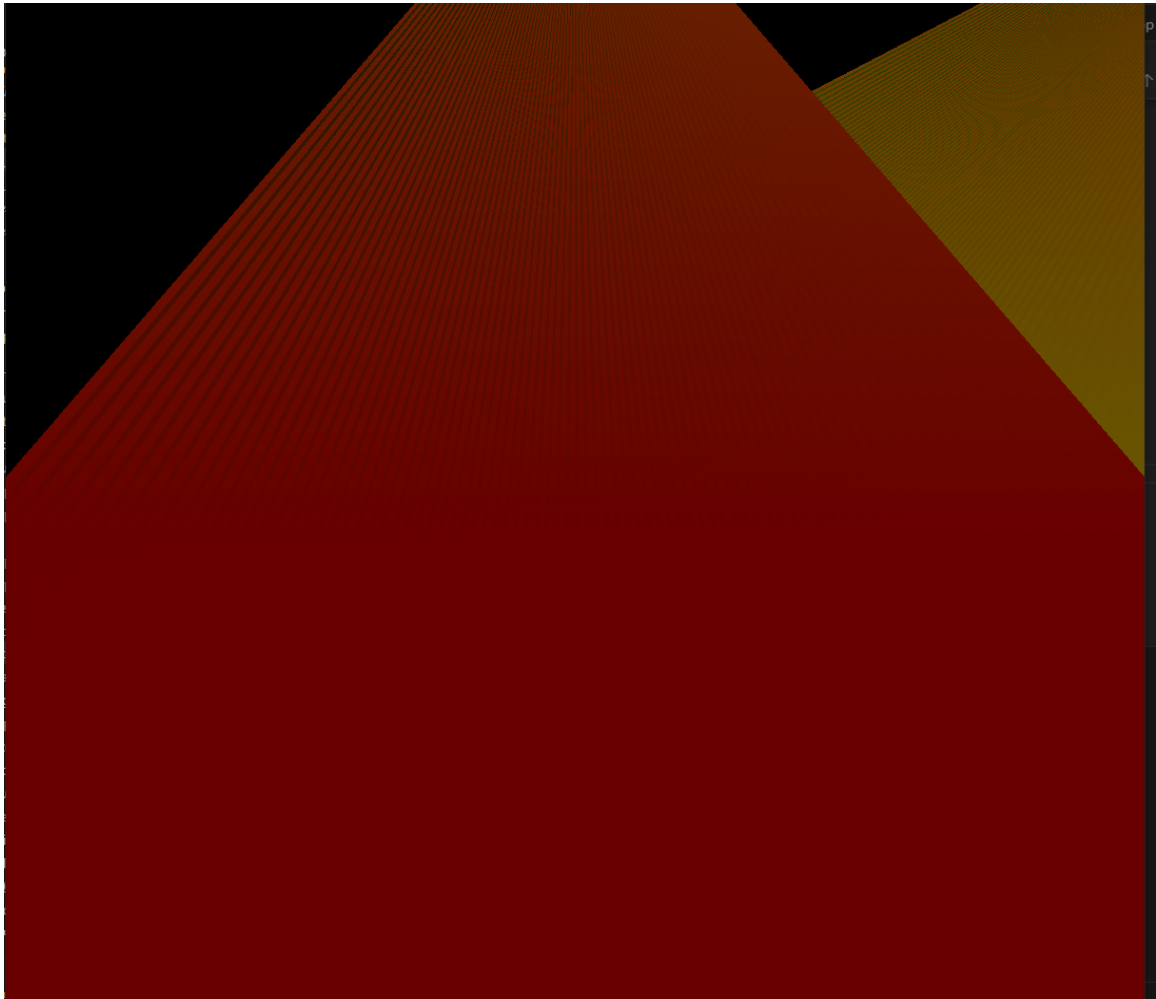
Input management primarily relies on a mapping in Figure 12 and handling the key bindings for different contexts. Each of the bindings will dynamically map to a function, with mappings able to be loaded and flushed to local JSON files for storage on startup and shutdown. A high-level input manager will also handle the mappings from contexts (different settings require different functionality) to input objects, which hold the relevant associations between keys and the functions.

### Non Euclidean Rendering Iterations

Throughout the development process, our team went through numerous iterations and testing phases to achieve accurate and consistent rendering of non-Euclidean shapes within our custom

engine. Early versions of the system struggled with issues such as distortion artifacts, incorrect projections, and performance bottlenecks. To resolve these challenges, we systematically refined our rendering pipeline, implementing and adjusting complex mathematical transformations such as the Weierstrass and Poincaré models to project hyperbolic geometry into a viewable space. Each iteration was informed by targeted tests, both visual and mathematical, to verify geometric fidelity and performance under different conditions. Through this iterative process, we were able to gradually correct inaccuracies, optimize rendering efficiency, and ultimately arrive at a solution that reliably visualizes hyperbolic spaces in real-time, laying the groundwork for both interactive gameplay and further experimentation.

### *Initial Iteration*

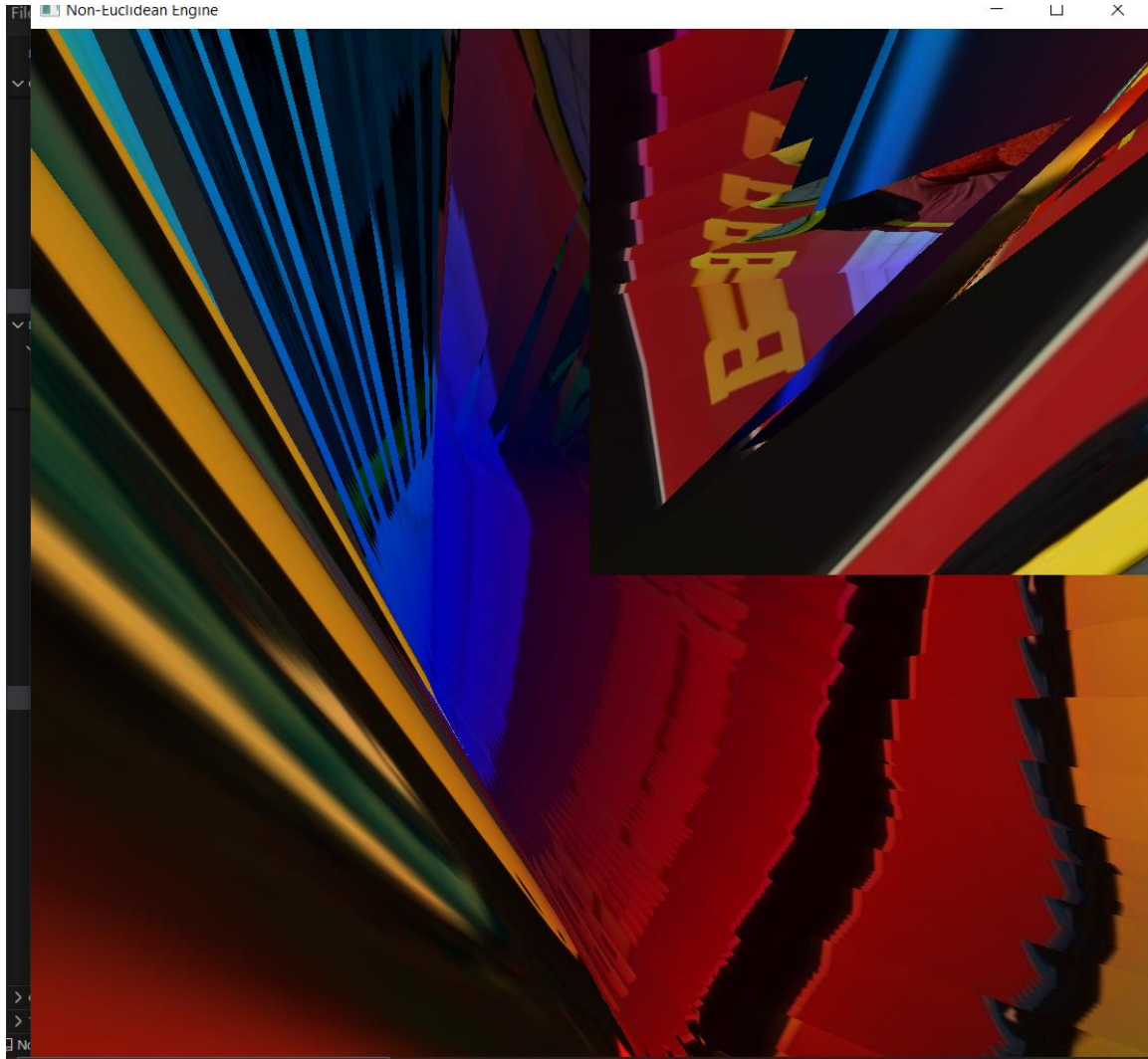


*Figure 21 - Initial Non-Euclidean Rendering Integration Test*

This image was one of the earliest attempts at rendering a  $\{4,5\}$  tile in our system using the custom resource management and system scheduler. This was after unit testing that rendering a  $\{4,5\}$  tile was possible, but was during the integration phase. It is obvious that there are multitudes of issues

with this current iteration. It was found that the buffers used to render the tile were not set properly, as well as the shader we were using was not fully correct.

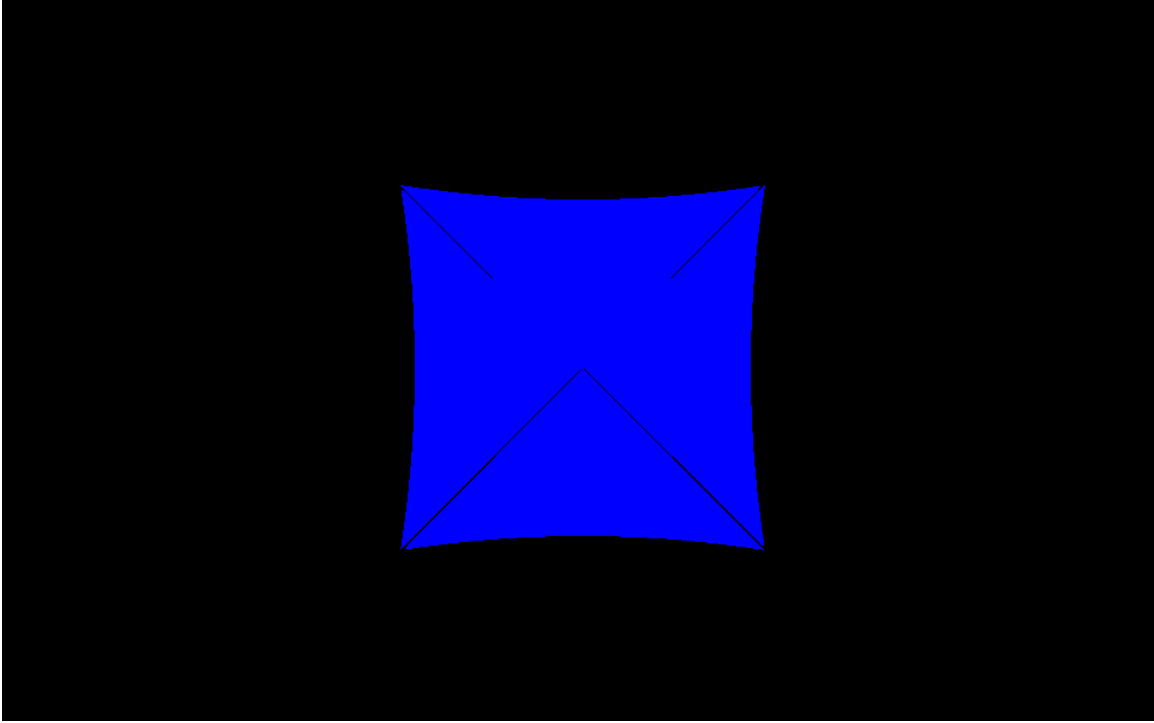
### *Second Iteration*



*Figure 22 - Second Integration Test*

This was a milestone achieved during the integration phase. It still was a long way away from the final solution, however, it was one of the first tests that produced an image in which you could see a sprite rendered with the  $\{4,5\}$  tessellation. The main issue with this iteration was that the UVs for the texture were being set improperly. UV mapping is the 3D modeling process of projecting a 3D model's surface to a 2D image for texture mapping.

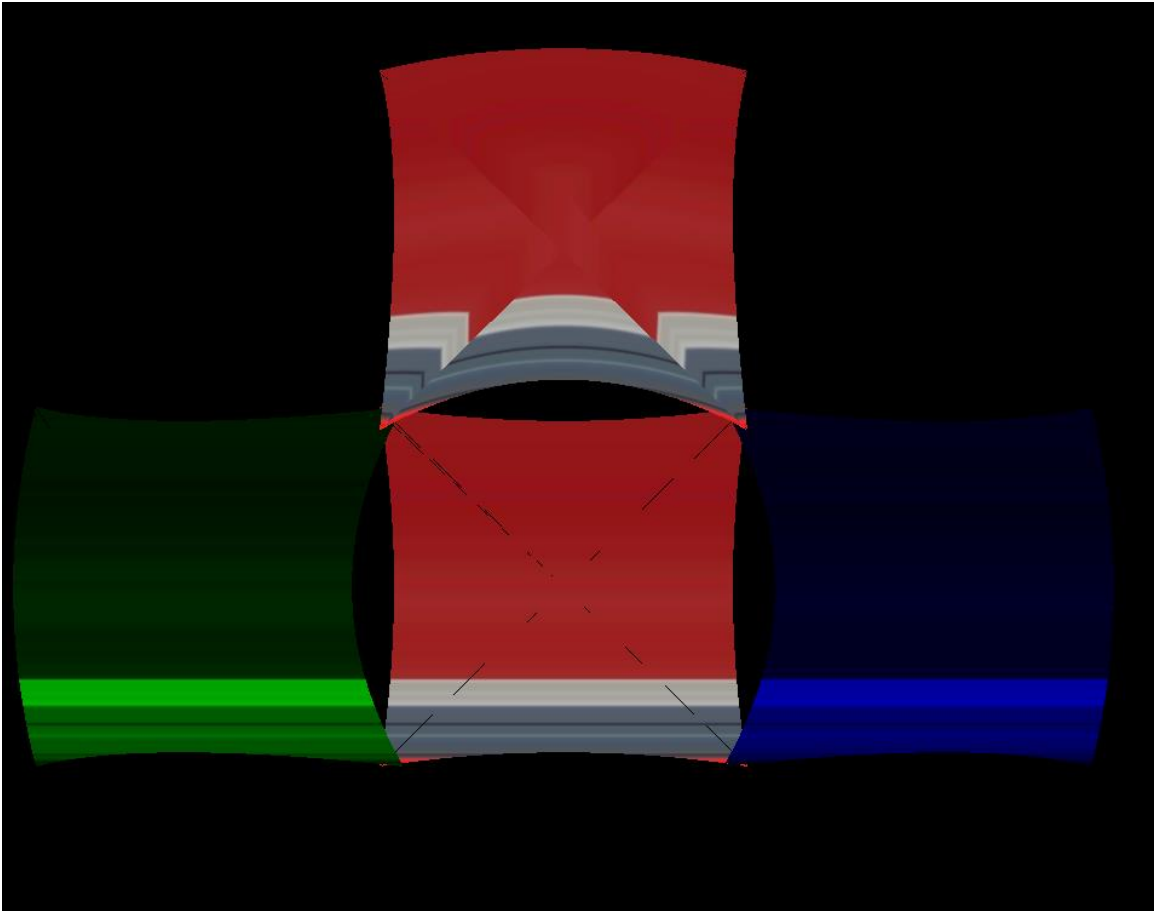
### *Third Iteration - Single Non-Euclidean Tile*



*Figure 23 - Initial PQ Tile Rendering in System*

This milestone was achieving a single PQ tile being properly rendered in the full system integration. At this step, texture mapping was still not working, but this was a major iteration due to it being proof that this system could do what was managed. There were still issues with this iteration, being that no textures were rendered, it was only a single tile, and this project needs support for multiple tiles, and there was an “X” that was not being rendered on the tile.

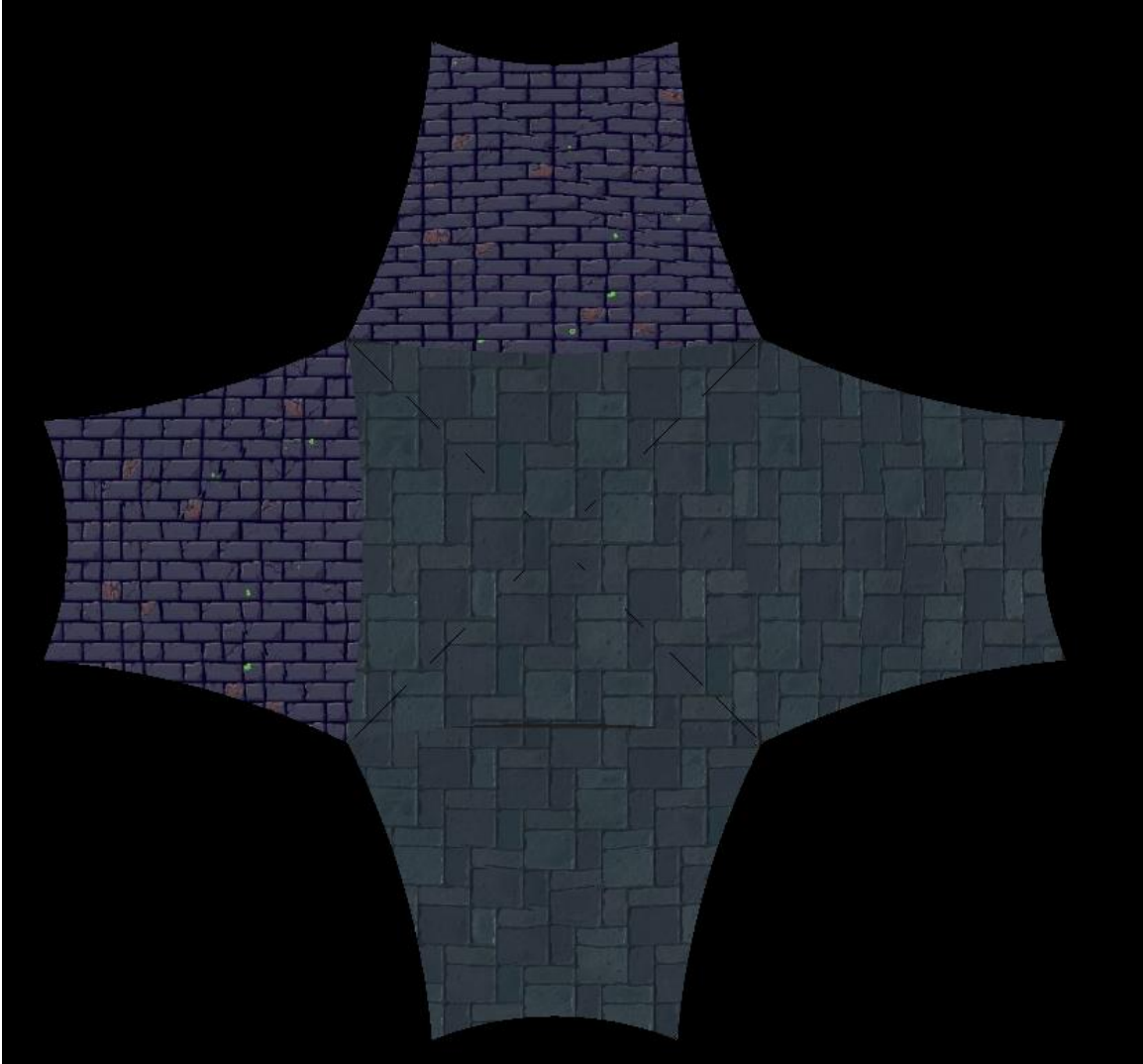
#### *Fourth Iteration - Multiple Non-Euclidean Tiles*



*Figure 24 - Multiple PQ Tiles Being Rendered*

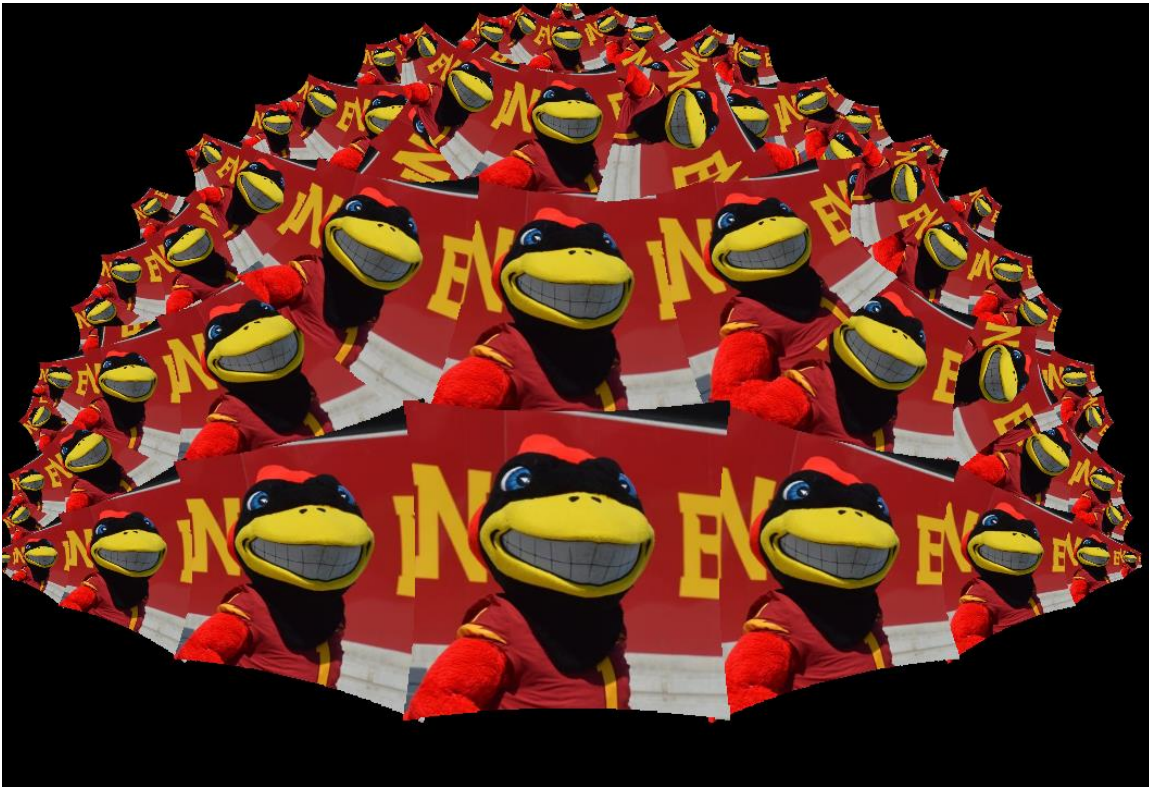
This iteration builds on the previous by rendering multiple tiles in a single frame. This iteration still had problems with texture mapping, but there was progress being made. The main issue found in this iteration was that the mapping of tiles on the screen differs from how mapping multiple tiles in a Euclidean space occurs.

*Fifth Iteration - Proper Placing & Texture Mapping*



*Figure 25 - PQ Tiles Being Properly Placed*





*Figure 26 - A Bunch of Cy's in a Hyperbolic Render*

This iteration was the last major iteration before the current state of the system. This iteration fixed 2 of the biggest issues still left for the rendered. It implemented the rotation matrix to properly place multiple tiles on the screen such that there is no overlap between tiles. It also finally corrected texture mapping onto a single Non-Euclidean tile. It can also be seen in Figure 26 that as the tiles get farther and farther away from the center tile, the texture becomes more distorted. The major issue with this iteration is that it was poorly optimized, so loading times were longer than expected. The current iteration fixes this problem.

## 6.3 DESIGN ANALYSIS

### Functionality That Works Well

The implemented design demonstrates a clear and compelling division between game design and engine development, facilitating parallel progress in both domains. One of the most successful aspects of the implementation is the prototyping and integration of core gameplay mechanics. Essential elements such as player movement, tile mapping, lighting, monster behavior, and inventory management have been implemented and tested thoroughly in Unity. Staged development—prototyping, integration, and scene creation—has proven effective in managing progress and ensuring alignment between functionality and intended gameplay. By concluding each stage with structured evaluations, the team was able to maintain project feasibility and adjust the timeline as needed.

Mechanics such as the A\* pathfinding algorithm for monsters, inventory-based shop systems, and the use of lighting as a gameplay constraint (particularly in the forest scene) all function as intended. These implementations not only reflect successful technical execution but also meaningful design decisions that enhance gameplay. The evidence for the success of these implementations is found in the working Unity demo, where the mechanics operate cohesively and provide a playable and immersive experience. Creating multiple interactive scenes, including the home farm and forest areas, further supports the system's effectiveness in unifying design concepts and functional code.

Similarly, the custom game engine showcases a solid foundation, with key subsystems—such as the Entity-Component-System architecture using the ENTITAS library, input management, and resource management—already in place and operational. The engine successfully renders a non-Euclidean tile map using a {4,5} hyperbolic tessellation, displaying over 3,000 tiles simultaneously. Implementing z-layered rendering allows for proper visual stacking of entities such as rocks and the player, contributing to the game world's clarity and depth. Performance optimizations, such as batching nearby tiles to minimize buffer allocations and draw calls, have shown promising results in maintaining engine performance during complex scenes.

The engine itself is functional with excellent structuring to allow for simple integration of new subsystems due to the scalability of the CMake toolchain. Furthermore, our general design decisions are all in accordance with recommended best practices from the external libraries we used, resulting in an overall well-designed and implemented project. Additionally, the API used to access and set up the engine for use in a game is relatively simple and intuitive to use with the App entry point. However, abstracting this into a compile-time system can assist in making the process even better for users and developers alike.

### Functionality That Does Not Work as Expected

Despite the successes, several features either did not function as intended or could not be completed within the project timeframe. The most notable limitations include the lack of integration between the Unity prototypes and the custom engine, and the incomplete development of some planned mechanics and scenes. Features such as NPC interactions, advanced trap mechanics, additional seed types, the slip-n-slide scene, and the blue forest area remain unimplemented. These gaps are primarily attributable to time constraints and the initially ambitious scope of the project. The absence of these elements limits the overall completeness and polish of the final product.

On the engine side, essential systems such as sound support, collision detection, and broader tessellation support are not yet in place. Although the core rendering and input systems are operational, these missing features hinder full gameplay implementation within the engine. The rendering pipeline, while functional, still requires optimization, particularly in how it handles non-Euclidean transformations and tile visibility determination. Moreover, while custom developer tools, such as texture atlasing and animation editors, have been developed, they may still require refinement to be fully production-ready.

The principal reason these aspects did not meet expectations is the misalignment between the timelines of the game design and engine development teams. The game design team, working within Unity, moved faster regarding prototyping and feature iteration, whereas the engine team focused on building lower-level systems that required more time and testing. A more synchronized

planning process and a more straightforward integration strategy between the two teams could have mitigated these issues.

## 7 Ethics and Professional Responsibility

The overarching team philosophy was utilitarianism. Our group aimed to do good work to increase our users' overall happiness. To do that, a solution that users will enjoy, use, and come back to is needed. Throughout this project, this group aimed to act professionally and respectfully to achieve that ethical philosophy.

### 7.1 AREAS OF PROFESSIONAL RESPONSIBILITY/CODES OF ETHICS

Area of Responsibility	Definition	Relevant Item from the Code of Ethics [8]	Teams' interaction with Area / Code of Ethics Item
Work Competence	Do the best work possible within our skill sets and timeline. Finish tasks on time and in a professional manner.	II.2.a. Engineers shall undertake assignments only when qualified by education or experience in the specific technical fields involved.	To satisfy this code of ethics, time was spent researching the topics of this project and learning the software required. This way, the team was qualified to produce quality work in the field of this project.
Financial Responsibility	The ability to create solutions within a reasonable budget and market those solutions at honest values.	II.4.c. Engineers shall not solicit or accept financial or other valuable consideration, directly or indirectly, from outside agents in connection with the work for which they are responsible.	This team is upholding this area of responsibility by using open-source resources and code bases. This reduces the budget of the proposed solution, as well as avoids using unnecessary services. This will lead to the cost of the project being smaller than if paid resources were used
Communication Honesty	Being honest about the work done and the state of the project to stakeholders	II.3.b. Engineers may express publicly technical opinions that are founded upon knowledge of the facts and competence in the subject matter	This team has interacted with this area when communicating with the advisor on this project. This team has put an effort to be honest about the progress made to our advisor to get valid feedback.

			Furthermore, this group strives to communicate our progress within the team honestly.
Health, Safety, Well-Being	An attempt should be made not to harm the well-being of stakeholders	I.1. Hold paramount the safety, health, and welfare of the public	This group holds this area of responsibility in high regard. The aim of this project is to develop a solution that does not harm the well-being of others and the public.
Property Ownership	Do not use the property and ideas of others without consent	III.9.b Engineers using designs supplied by a client recognize that the designs remain the property of the client and may not be duplicated by the engineer for others without express permission.	The team has interacted with the area of responsibility regarding our code base. The group has chosen to write our own code base and not copy and paste others' code from other sources.
Sustainability	Develop products in a way that it does not harm/does minimal harm to the environment.	III.2.d Engineers are encouraged to adhere to the principles of sustainable development to protect the environment for future generations.	This code of ethics was not a guiding factor in the project's determination. Due to the low dependency on hardware and software costs, sustainability was not a high area of concern this semester for the team.
Social Responsibility	Develop solutions that attempt to improve the world	III.2.a Engineers are encouraged to participate in civic affairs; career guidance for youths, and work for the advancement	By striving to create a good and fun product, the hope is to provide users worldwide with enjoyment.

*Table 10 - Areas of Professional Responsibility*

#### One Area Team Performed Well: **Financial Responsibility**

The team has performed this area, as seen how this project stuck to a low-cost budget. Many software projects often have bloated budgets and set unrealistic expectations for personnel hours.

The bloated budgets usually cause the cost of developing software to be high, which leads to the cost of buying software to be high as well. The team's approach to not having a bloated budget is to avoid using services that explicitly require payments. One example of this is Unity Version Control. Given the size of the project, this group would have to pay to use that resource monthly. Instead, the group decided to use git as a version control history. The tradeoff is that managing game design files will be more difficult, with the benefit of reduced cost. Another way this team reduced the budget is by using open-source software and resources. The game engine is developed in OpenGL, which is free software. A choice could have been made to develop software that costs money. The benefit of choosing paid software is that it comes with extra resources. However, that benefit is not worth the cost of the software itself. This upheld the ethic because using low-cost technology and resources allows the project's budget to be well-maintained.

#### One Area Team Lacked: **Social Responsibility**

An area of responsibility this group did not focus heavily on is social responsibility. The design of this project left a low impact on improving the social environment. This is due to the fact that the primary purpose of this project is for personal enjoyment. This project aimed to bolster an individual's enjoyment, but not a society or community. There was an attempt to make the game engine usable for the academic and research community. However, this was not achieved due to the time constraints and focusing on the primary user needs. Given more time, it would be reasonable to develop an API that would allow for mathematics and other research to visualize models in hyperbolic space.

## 7.2 FOUR PRINCIPLES

Broad Context Area	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	Promotes cognitive development and mental well-being through engaging gameplay.	Minimizes safety risks by avoiding harmful or exploitative content.	Offers players choices in gameplay to explore and learn at their own pace.	Ensures accessibility features for diverse populations, promoting inclusivity.
Global, cultural, and social	Encourages global appreciation for abstract concepts and STEM education.	Avoids cultural insensitivity or conflict by respecting diverse values.	Respects players' cultural identities by avoiding stereotypes or exclusive designs	Provides equitable access regardless of nationality or cultural background.
Environmental	Reduces energy consumption by optimizing computational efficiency.	Minimizes negative impact by designing for broad hardware compatibility.	Allows users to choose energy-efficient settings for gameplay.	Promotes sustainability by demonstrating responsible resource usage.
Economic	Creates jobs and promotes STEM learning for long-term economic growth.	Reduces potential financial strain on consumers through affordability strategies.	It provides options for consumers to purchase within their financial means	Makes the product accessible across income levels by tiered pricing or discounts.

*Table 11 - Four Principles*

Important Broader Context-Principle Pair: **Public Health, Safety, and Welfare - Beneficence**

This project aimed to foster mental well-being and cognitive development by providing an engaging and thought-provoking experience. By incorporating features like educational tools and problem-solving mechanics, this group ensured the game delivers positive, meaningful impacts to players. To achieve this, focus was placed on balancing entertainment with learning and conducting user testing to refine how the game benefited users cognitively.

Lacking Broader Context-Principle Pair: **Economic - Beneficence**

The project has a limited effect on long-term economic growth and STEM learning. This is due to the fact that the project will be released as free open-source software and will not generate income to create jobs. The team can improve our economic benefits by creating a quality open-source engine that companies would like to reuse. Furthermore, the creation of educational material on the engine would be able to promote STEM learning.

## 7.3 VIRTUES

Three virtues essential to the team

- Honesty
- Cooperativeness
- Responsibility

These virtues were decided as the most important to our group as a team. Honesty is an important virtue to have when working as a group. Individuals must be honest with each other when communicating what has been done and what concerns they have. As a team, honesty is essential when communicating our plans for this project to the outside world and our advisors. To support this virtue, this team has been honest with each other about what has been done regarding work. An effort was made to be honest with our advisor about the evolving state of our project. Cooperativeness was vital to this team as well. Being a team, members must work together to solve problems and design solutions. One person cannot do this project, so teamwork must happen. This team has used weekly meetings to ensure everyone is communicating and on the same page about what is happening. There is open communication between what each team member is doing and open communication between exchanging ideas and troubleshooting. Finally, responsibility is an essential virtue of this project. Each team member must take responsibility for the tasks that they are given. They must take charge of their work and complete it to a high standard and on time. If this team is not being responsible, the deadline will not be met for this project.

### Individual Virtues

Tasman Grinnell

Virtue Demonstrated: **Communication**

One of the most significant responsibilities that the team has assigned to me has been to keep the team on track and communicate to ensure that the team is on track, specifically with the plethora of assignments that we have for Senior Design. The reason why this is important to me is that keeping on top of deadlines is generally important for me, and being able to help keep the entire



team on track with my mindfulness of deadlines is a very important way to keep the team working well.

Virtue Lacking: **Clear and Thorough Documentation**

The virtue of clear (and thorough) documentation is a very important virtue for me, due to the fact that documentation is one of the most essential facts in creating software that helps extend longevity. By providing documentation, others can continue work and iterate on versions after the initial teams move on from the project. Additionally, documentation is incredibly important in industry to allow for the smooth functioning of services and software, resulting in documentation being very useful. To demonstrate this, we can begin to generate our own documentation for the software using various services and documentation creating tools.

Joshua Deaton

Virtue Demonstrated: **Helpfulness**

A virtue that I have demonstrated is my willingness to help others. I have demonstrated this by creating the toolchain that allows my team to perform development. Furthermore, I have improved upon this build chain from the suggestions of Ben and Tasman, as well as helping Ben and Tasman if they were ever having difficulties adding in new libraries or trouble compiling. This virtue is important to me as being a helpful person enables the team to be more successful and productive.

Virtue Lacking: **Consistency**

One virtue I have lacked to demonstrate as much as I would like throughout this project was consistency. Much of the work I have done is done in bursts and is not consistent. Part of this is due to the numerous other personal projects, class projects, and papers. By failing to be consistent, it can affect the team's work ethic as well as the quality of our final product. I will take this experience and see how not being consistent can prohibit a project's success. I hope to improve upon this virtue by going above and beyond my responsibilities in my future endeavors.

Lincoln Kness

Virtue Demonstrated: **Willingness to Take Initiative**

A virtue that I have demonstrated is a willingness to take ideas/prototypes to consolidate them into one. I have demonstrated this throughout a few phases of our project when it comes to our prototypes. What I mean by this is that we would individually work on implementing one specific game mechanic in Unity. I have taken the time to take each of the prototypes made and combine them into one package, allowing the different prototypes to work together.

Virtue Lacking: **Thorough Documentation**

One virtue that I seem to have struggled with this project was documentation. This is important because it allows others in the group to easily be able to look at the work I have done and be able to understand it easily. A way that I can work to start demonstrating this virtue is to create more in-depth documentation, whether that means creating more concise comments in code or creating separate documents explaining things in a broader sense. For my future work, I hope to make strides in showing these virtues more.

Ben Johnson

Virtue Demonstrated: **Problem Solving**

Throughout this project, I have taken a proactive approach to solving problems that arise. In group environments, it is important to solve issues quickly and effectively. Otherwise, small snags might hold up the entire team for a long time. I have done my best to help teammates with problems as they arise, and also to reach out for help when I get stuck.

Virtue Lacking: **Organization**

When working on large projects in groups, it is important to create a project structure to organize and coordinate team efforts. For a software project like this one, organization mostly entails creating an issue board and dividing up those tasks among team members over a set timeline. I didn't get one of these setups until halfway through the semester, and even now it isn't used as effectively as it should.

Zach Rapoza

Virtue Demonstrated: **Keanu**

One virtue I have modeled this project on was keanu, which, based on the slides, means being cool and easygoing. This is important to me because while you need to take things seriously, if there is one thing I have learned throughout college, it is that I am not perfect. As such, it is important to understand that sometimes you just have to go with the flow. One way that I have demonstrated this virtue is by trying to be more laid back during meetings. This is not to the extent that I zone out, but rather let other people get their full idea formed and out on the table before I start seeing if it will hold up.

Virtue Lacking: **Responsiveness**

One virtue that I have struggled with is responsiveness. This is important because you need to respond in a quick and timely manner. This ensures that people do not end up with a roadblock that they cannot overcome. One way I have demonstrated this is by prompt responses to communication on weekends and always responding by the end of the day. One way in which I have failed to do this is when I am really busy, I will put off responding for a couple of hours until after I have had a chance to breathe.

Spencer Thiele

Virtue Demonstrated: **Cooperativeness**

One virtue I believe I have demonstrated is cooperativeness. Cooperativeness is an important part of game design and development because of how interconnected the elements of a game are. When leading the brainstorming and design meetings, I took measures to ensure everyone gets a chance to come up with and communicate ideas through various techniques. This ensures we have the largest pool of ideas to draw from and build into our game. I also encouraged quick communication when stuck on bugs by implementing a bug bounty system.

Virtue Lacking: **Timeliness**

One virtue I have been lacking throughout this project was Timeliness. In a team development setting, delivering project pieces on time is important if you want to keep to the planned schedule. While I did reply to communications swiftly, I wasn't finishing my project work in the timeframe I promised. Running into unpredictable issues is a normal part of software development. Still, I failed to deliver a finished prototype on time multiple times simply due to a lack of invested hours. Going forward, I will allocate more time to development for future projects.

Cory Roth

Virtue Demonstrated: **Thoroughness**

One virtue that I have demonstrated well in this project was my ability to be thorough and attentive to the project. I have been tasked with taking the notes during meetings and ensuring that the group has all the information it needs to succeed. I have attended all the meetings and lectures, been attentive, and taken good notes for each meeting and lecture. This allowed for the group to finish assignments quicker and to a higher standard because we were able to look back on what we did and were logging information. I have been thorough because I have been making sure that the assignments we submit are of quality and thought.

Virtue Lacking: **Time Management**

One virtue that I feel I have not demonstrated well is my time management. I have had a busy schedule with other classes, so I have not been able to commit as much time to this project as I would have liked to. This is important to me because in order to create a good project, time needs to be put into it. You need to put time into the work in order to be proud of it.

# 8 Conclusions

## 8.1 SUMMARY OF PROGRESS

### Game Engine

- Built up the engine from scratch with the majority of functionality necessary for game development
- Includes Input Handling, System Scheduling, Rendering Loop, Non-Euclidean Shaders, Tilemaps, etc.
- Offline Storage
- Custom Developer Tooling:
  - Texture Atlasing
  - Key Mapping Management
  - Animation Editor
- Custom Tile Mapping (Non-Euclidean and Euclidean Compatible)

### Game Design

- The design and iteration process was repeatedly undergone using a Unity Prototype.
- Playtesting to receive feedback on gameplay mechanics before transitioning to our engine
- Developed a fully fleshed-out game design
  - World Environment
  - Biomes
  - NPCs
  - Mechanics
  - Story

Throughout the project, our team made substantial progress in game engine development and game design. The project was divided into two major tracks: building a custom game engine from the ground up, and designing and iterating on gameplay systems using a Unity-based prototype. This bifurcated approach allowed us to make parallel advancements, validate gameplay mechanics early, and construct a solid technical foundation for future integration.

On the engine side, the team successfully developed a custom game engine that includes a wide range of core systems essential for 2D game development. These systems include input handling, system scheduling based on an Entity-Component-System (ECS) architecture, a real-time rendering loop, and support for advanced features such as non-Euclidean shaders and tile map rendering. Of particular note is implementing a rendering system capable of visualizing  $\{4,5\}$  hyperbolic tilings within the Poincaré disk model, allowing the engine to display complex, infinite-style geometry. The engine also supports multi-layered z-depth rendering, efficient instanced drawing, and configurable texture atlases. These accomplishments demonstrate both technical competence and a commitment to innovation within the project scope.

Concurrently, the game design team employed Unity as a rapid prototyping tool to explore and iterate on gameplay mechanics. This approach facilitated a cycle of design, playtesting, and refinement that helped ensure that the mechanics under development would be engaging and coherent when later ported to the custom engine. This process developed and validated several

core features, including A\* pathfinding for enemy AI, a player inventory and shop system, lighting-based exploration mechanics, and modular scene construction. Playtesting sessions provided valuable feedback that influenced balancing decisions and interface adjustments, allowing the game to evolve in response to user experience.

Together, these two parallel efforts advanced the project toward its core goal: to create a cohesive and technically innovative game experience grounded in custom-built tools. While the final deliverable includes a functional Unity demo and a robust, partially integrated engine, some features remain in development or unimplemented, such as NPC interactions, additional biomes, and full collision and audio systems in the engine. Nevertheless, the accomplishments thus far have laid a strong foundation, and the results demonstrate both feasibility and forward momentum. In summary, the project achieved many of its core objectives. The custom engine showcases original engineering in non-Euclidean rendering and engine systems, while the Unity prototype validates the core gameplay vision through hands-on testing and iteration. These results reflect a high level of collaboration, adaptability, and ambition from all team members.

## 8.2 VALUE PROVIDED

The design of our game and custom engine was developed with specific user needs and project goals in mind: to create a unique gameplay experience grounded in hyperbolic geometry, and to support this experience with a flexible, performant, and extensible custom-built engine. The project addresses both the experiential needs of the player—providing novelty, challenge, and immersion—and the technical challenges of visualizing and interacting with non-Euclidean spaces.

From a user experience standpoint, using  $\{4,5\}$  hyperbolic tiling's introduces a novel spatial structure rarely explored in mainstream games. This satisfies a demand for originality and cognitive engagement among players seeking unique environments and navigation challenges. Playtesting of our Unity prototype offered preliminary validation of this value: players consistently expressed curiosity and intrigue at the unfamiliar geometry, and many cited the exploration mechanics as the most memorable feature of the experience. These reactions suggest that the core design resonates with players looking for games that challenge their spatial reasoning and provide novel aesthetic experiences.

On the technical side, the project sought to overcome limitations of existing game engines by creating one that supports efficient rendering and manipulation of hyperbolic space. Mainstream engines like Unity or Unreal are not optimized for non-Euclidean rendering; they often require workarounds or custom shaders that are difficult to integrate cleanly. Our custom engine addresses this problem directly through native support for Weierstrass-to-Poincaré projection in the vertex shader pipeline and instanced rendering techniques tailored to hyperbolic tile maps. This approach has yielded a rendering system that can display thousands of tiles at interactive frame rates, which would be computationally expensive or infeasible using traditional Euclidean assumptions. Thus, the engine provides a concrete technical value: it fills a gap in current tooling for developers interested in non-Euclidean game design.

This project contributes to a growing interest in mathematical aesthetics and educational gameplay in the broader context of game development and procedural content generation. By demonstrating that hyperbolic space can be rendered and navigated efficiently, our work paves the way for further exploration of mathematical games, educational simulations, and spatially experimental narratives. For example, our engine could be adapted to teach geometry or topology in interactive, visual ways that go beyond static diagrams or textbook problems.

Moreover, the modular structure of both the gameplay systems and the engine components provides room for scalability and future development. Systems like ECS-based scheduling, tile map parsing, and dynamic lighting are all designed to be reusable and extendable. This makes the project not only a functional prototype but also a foundation for future research and creative development in the space of mathematically informed game design.

## 8.3 NEXT STEPS

### Summary of Next Steps

- Polish Engine
  - Performance Optimization
  - Intuitive API Design
  - Improve Non-Euclidean Map Generation
- Continue Integrating Unity Demo with Engine
  - More Areas
  - Flesh out the in-game story
- Finish / Continue Non-Euclidean Shaders
  - Add other tessellations

While this project marks a substantial milestone in the development of a custom engine and game prototype set in hyperbolic space, there remain several critical avenues for future work that would significantly enhance the functionality, usability, and creative potential of the system. These next steps would build directly upon the foundational accomplishments of our current engine and gameplay design.

A key area for continued development is engine polish and performance optimization. Although the engine currently supports interactive rendering of non-Euclidean environments, there is significant room for improvement in computational efficiency, especially for larger-scale maps and more complex shaders. Optimizing data structures, refining the rendering pipeline, and profiling memory usage would allow for smoother performance on a broader range of hardware.

Another priority is refining the engine's API to be more intuitive for future developers. As it stands, the engine is powerful but primarily suited for internal use by the original team. By formalizing documentation and improving the modularity and clarity of the API, the engine could become a viable platform for other designers and developers interested in experimenting with non-Euclidean geometry or custom rendering techniques.

On the gameplay side, expanding the non-Euclidean map generation system will be essential for creating more prosperous and more dynamic worlds. This includes support for procedural generation, additional tessellation types beyond the current {4,5} tiling, and improvements to how

map data is stored, loaded, and modified at runtime. Each of these steps would contribute to a more scalable and varied gameplay experience.

One of the most important long-term goals is the continued integration of the Unity prototype with the custom engine. The Unity version of the game currently functions as a testbed for gameplay mechanics, while the custom engine handles rendering and spatial logic. Unifying these two systems—or translating the full gameplay into the custom engine—will allow for a seamless and cohesive development process, reducing redundancy and enabling features like real-time debugging, cross-platform builds, and tighter iteration loops.

In addition to technical improvements, there is also creative and narrative work to be done. Expanding the in-game story, designing new gameplay areas, and completing the development of the non-Euclidean shaders will help deliver a more immersive and complete experience to players. These elements are essential for transforming the current prototype into a polished, fully-realized game.

Finally, there is potential for entirely new projects based on what we have accomplished. For example, the engine could serve as the foundation for educational tools that visualize hyperbolic geometry, or for creative tools that enable artists and designers to explore novel spatial configurations. In a broader societal context, such projects could foster public engagement with advanced mathematics and spatial reasoning, bridging the gap between abstract theory and interactive experience.

## 9 References

- [1] IEEE Standards Association, "IEEE Standard 2983: Title of the Standard," [Online].IEEE Available: <https://standards.ieee.org/ieee/2983/10523/>. [Accessed: Nov. 19, 2024].
- [2] "IEEE Guide for Software Quality Assurance Planning," in IEEE Std 730.1-1995 , vol., no., pp.1-20, 10 April 1996, doi: 10.1109/IEEESTD.1996.80817.
- [3] IEEE Standards Association, "IEEE Standard 12207: Systems and Software Engineering – Software Life Cycle Processes," [Online]. Available: <https://standards.ieee.org/ieee/12207/5672/>. [Accessed: Nov. 19, 2024].
- [4] Rogue Temple, "Hyperrogue Development Page," [Online]. Available: <https://www.roguetemple.com/z/hyper/dev.php>. [Accessed: Nov. 19, 2024].
- [5] D. Osudin, C. Child, and Y.-H. He, "Rendering Non-Euclidean Space in Real-Time Using Spherical and Hyperbolic Trigonometry," in Computational Science – ICCS 2019, vol. 11539, Springer International Publishing, 2019, pp. 543–550. doi: 10.1007/978-3-030-22750-0\_49.
- [6] Rogue Temple, "Hyperrogue Models Page," [Online]. Available: <https://www.roguetemple.com/z/hyper/models.php>. [Accessed: Nov. 19, 2024].
- [7] M. Bremer, "Non-Euclidean Geometry Explained," Hyperbolica Devlog #1 [YouTube video]. [Online]. Available: <https://www.youtube.com/watch?v=19o8.01742>. [Accessed: Nov. 19, 2024].
- [8] NSPE, "Code of Ethics for Engineers," National Society of Professional Engineers, [Online]. Available: <https://www.nspe.org/resources/ethics/code-ethics>. [Accessed: Nov. 19, 2024].
- [9] M. Bacarella, "EnTT: Gaming meets modern C++," GitHub repository. [Online]. Available: <https://github.com/skypjack/entt>. [Accessed: May 2, 2025].
- [10] D. Herberth, "GLAD: Multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator," GitHub repository. [Online]. Available: <https://github.com/Davidde/glad>. [Accessed: May 2, 2025].
- [11] GLFW Project, "GLFW: A multi-platform library for OpenGL, OpenGL ES, Vulkan, window and input," GitHub repository. [Online]. Available: <https://github.com/glfw/glfw>. [Accessed: May 2, 2025].
- [12] O. Cornut, "Dear ImGui: Bloat-free Graphical User Interface for C++ with minimal dependencies," GitHub repository. [Online]. Available: <https://github.com/ocornut/imgui>. [Accessed: May 2, 2025].
- [13] G. Guennebaud, B. Jacob, and others, "Eigen v3," GitLab repository. [Online]. Available: <https://gitlab.com/libeigen/eigen/>. [Accessed: May 2, 2025].
- [14] C. Riccio, "OpenGL Mathematics (GLM)," GitHub repository. [Online]. Available: <https://github.com/g-truc/glm>. [Accessed: May 2, 2025].
- [15] N. Lohmann, "JSON for Modern C++," GitHub repository. [Online]. Available: <https://github.com/nlohmann/json>. [Accessed: May 2, 2025].



[16] Sean T. Barrett, "stb: Single-file public domain libraries for C/C++," GitHub repository. [Online]. Available: <https://github.com/nothings/stb>. [Accessed: May 2, 2025].

# 10 Appendices

## APPENDIX 1 – OPERATION MANUAL

### Engine Manual

#### Engine Overview

The game engine repository uses the following organization:

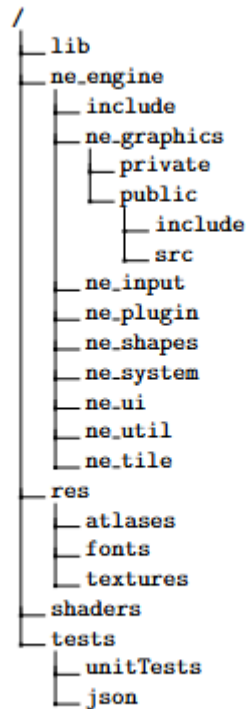


Figure 27 - Appendix 1: Outline of Engine Repo Structure

**lib/** contains all the libraries required for functionality;

**ne\_engine/** contains the source code for each submodule and core functionality

**res/** contains atlases, textures, and fonts for use in the engine (drawing sprites, displaying text, etc.);

**shaders/** contains the shader code for the Non-Euclidean shaders used by the Graphics Processing Unit;

**tests/** contains C++ files used for testing functionality, using a high-level version of the game engine rendering loop;

**tests/unitTests** contains Catch2 unit tests.

### Using the Engine

For development or use of the game engine, install the prerequisites for building and interfacing:

1. Git
2. CMake
3. Ninja
4. Catch2
5. Initialize Submodules

Before using the engine itself, make sure that CMake and Ninja are both on your PATH environment variable. This will allow for the commands for generating the build itself.

### Development

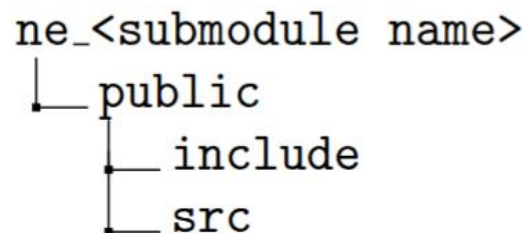
To set up the repository, follow the instructions below after installing the prerequisites:

1. Clone the Repository.
2. Navigate to the Repository's root directory (**NonEuclideanEngine/**) and update and initialize the submodules using `git submodule update --init`
3. Initialize project using the command `cmake -S tests -G Ninja`
4. Build the project using the command `cmake --build build`
5. Run the compiled executable.

The CMake file found in the **tests/** folder specifies the target files to compile, and the **tests/unitTests/** folder is used for creating and specifying unit tests used for testing functionality.

### Coding Conventions:

When adding functionality, a submodule will have the following structure:



*Figure 28 - Appendix 1: Submodule Convention*

Note: When adding classes or functionality, the **CMakeLists.txt** file in the **ne\_engine/** directory needs to be updated to include the additional files or subdirectories for building. Additionally, the **ne\_engine.hpp** file found in **ne\_engine/include/** folder needs to be updated as well.

For each submodule, the **include/** directory hosts C++ Header files (with file extension **.hpp**) while the **src/** directory holds the C++ source files.

### *Using the Engine*

Ensure that all prerequisites are met. In your repository, clone the engine repository and add the directory as a subdirectory (e.g., in your root directory, create a `CMakeLists.txt` file and include the command `add_subdirectory(NonEuclideanEngine)`).

Adding the engine repository as a subdirectory will allow for full engine functionality through a single include for your files (`#include "ne_engine.hpp"`).

### *Custom Components*

Interaction with the engine is performed through the `App.hpp` class, which is used for adding systems and initializing global resources.

To add a component, the `App.InsertResource<Type T>()` function must be called, where `Type T` is a class that needs to be registered with the engine.

```

ImGui::CreateContext();
App()
    .InsertResourceBase<Window, GLFWWindow>(1080, 1080)
    .AddPlugin<DefaultPlugins>()
    .AddPlugin<WorldPlugin>()
    .InsertResource<TextureManager>()
    .InsertResource<TileMap>()
    .InsertResource<Camera>(camera_pos, camera_up, proj_mat)
    .InsertResource<Input>()
    .Run();

glfwTerminate();
return 0;

```

Figure 29 - Appendix 1: Example Inserting Resources

Figure 29 is an example of inserting resources into a program using the system. This example uses for main resources outside the base resource Window. TextureManager is a system to manage loading textures, and TileMap manages the location of tiles and where they should be rendered. The Input resource is for binding keyboard inputs into updates.

Below is an example of a Custom Component

```

benjisu, 4 weeks ago | 2 authors (benjamin johnson and one other)
class TextureManager {
public:

    TextureManager() { stbi_set_flip_vertically_on_load(true); }

    Result<std::nullptr_t, std::string> loadAtlas(const char* path);
    std::optional<AtlasedTexture> getTexture(const std::string& name);

    Result<std::nullptr_t, std::string> LoadTextures(const char* path);

private:
    std::unordered_map<unsigned int, Texture> _atlases;
    std::unordered_map<std::string, AtlasedTexture> _textures;
};

```

Figure 30 - Appendix 1: Texture Manager Interface

```
static void
LoadTextures(Resource<TextureManager> texture_manager)
{
    texture_manager->LoadTextures("../res/textures");
}
```

Figure 31 - Appendix 1: Load Textures Example

```
auto entity = registry.create();
auto texture_result = texture_manager->getTexture(up_tile.sprite);
if (texture_result)
{
    AtlasedTexture texture = texture_result.value();
    rotated_up_tile.recalculate_uvs();
    registry.emplace<AtlasPQtile>(entity, rotated_up_tile, texture, 0.0f);
}
```

Figure 32 - Appendix 1: Use Texture Example

Figure 30 depicts the interface designed for a texture manager. This texture manager gets loaded with textures on startup of the app using the function defined in Figure 31. This abstraction relieves the developer from having to ensure every texture they want rendered gets loaded properly. Figure 32 shows an example usage of that manager. The program calls `getTexture` on the manager and passes in the sprite name it wants. If that sprite is in the texture manager, it will return true. From that, the developer can create an `AtlasedTexture` object from the results, which will have all the proper data relating to the texture loaded. The texture can then be added to the registry, linked to a specific tile it wants to be rendered on.


## Game Manual


### Setup

The game has two main components: a completed, working build via Itch.io and the development environment in Unity. The setup for each is as follows.

Itch.io: To install and run the Itch.io build of the game, follow the procedure below

1. Navigate to [Lights Out](#) and download the zip file corresponding to your OS
  - a. We STRONGLY suggest using a Windows system due to issues on Linux
2. After the download completes, unzip the file and open the folders until you find the executable file corresponding to your system
  - a. Windows
 

 Farming Prototype.exe	4/22/2025 4:43 PM	Application	651 KB
---	-------------------	-------------	--------
  - b. Linux
 

 LightsOutLinux.x86_64	4/22/2025 4:58 PM	X86_64 File	15 KB
---	-------------------	-------------	-------
3. From there, run the executable file, and you are good to go
4. Play the game and enjoy

5. Look at the Playtest Document in Appendix 3 for playtest instructions

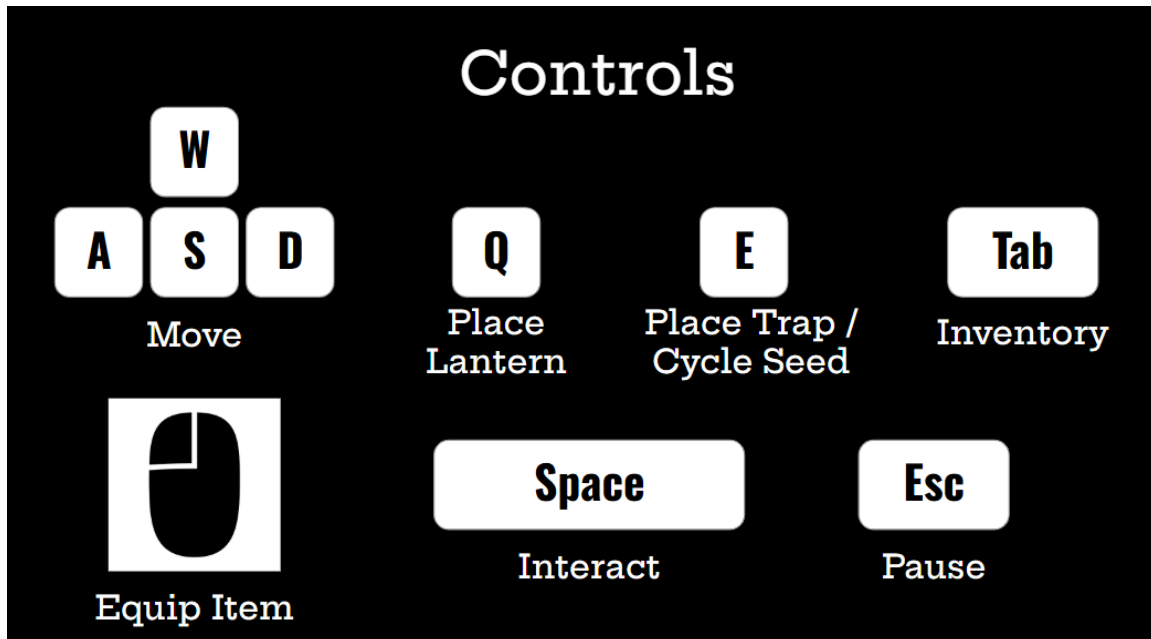


Figure 33 - Appendix 1: Controls Mapping

Unity: To set up the Unity development environment, follow the procedure below

1. Navigate to <https://unity.com/unity-hub> and download Unity Hub
2. In Unity Hub, navigate to Install and install the Unity editor (Unity 2022.3.49f1)
3. Next, we need to clone our development repo (<https://github.com/sdmay25-37/GameDesignUnity>) using either Git or by downloading the zip
  - a. You can install Git via <https://git-scm.com/downloads>
4. To load the project, navigate to the Projects tab in Unity Hub and select Add Project. Select the from disk option from there and locate the path to the cloned repository.
5. You will also need to have an IDE for C# installed on your device
  - a. We recommend using VS Code – <https://code.visualstudio.com/download>

#### Development (Unity Only)

To modify the game, you should follow the general practices for development in Unity. Our game follows the general structure of Unity games, consisting of a combination of game objects and their corresponding scripts.

To modify positioning of game objects, navigate to the corresponding scene (the scenes are independent of each other, i.e., what you do to the game objects in one will not affect the other), and then using the scene tools you can resize, relocated, and modify the game objects that you require.

To modify the scripts, navigate to the corresponding subfolder in scripts (i.e., enemies, farmer, or inventory) and from there click on the script to open it in your C# IDE, and go crazy.

### Testing (Unity Only)

Following any modifications that you make, you will need to test them. To do so, navigate to the Unity editor, wait for the update to load, and then select the run button in the top right corner. Following that, check the console for any errors and then check that your changes took effect and are bug-free.

## APPENDIX 2 – ALTERNATIVE/INITIAL VERSION OF DESIGN

### Initial Engine Design

The initial design of the game engine was not very different from the current version, only with two significant changes:

1. Entity-Component-System (ECS) Software;
2. Repository Organizations.

For the ECS, a custom version was created, but multiple failures in terms of performance combined with manpower forced a pivot to a well-established ECS system, ENTT. The initial version was simply impossible to prepare and finish due to the time constraints and scope of the project. Considering that ENTT and other ECS libraries have been in development for multiple years, this was not very surprising.

Additionally, as our repository increased in size and functionality, the number of subfolders was increased to enhance readability and compartmentalization. Figure A2.1 details a high-level overview of the changes that were performed.

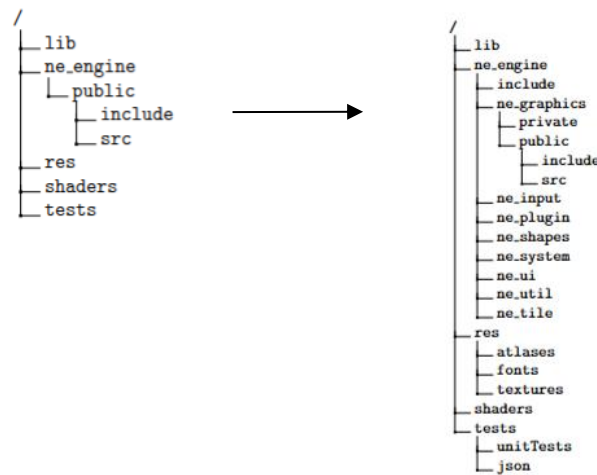


Figure 34 Appendix 2: Repo Structure Changes

### Initial Game Design

Since the game design process was broken down into two separate states, Unity development and porting over to the Non-Euclidean engine, they will be handled separately.



For the Unity prototype, the initial game's base functionality follows closely what is currently implemented in the play test build, with the only differences being:

1. Reduction in the number of scenes implemented
2. Not implementing the extra “planned” game mechanics of fishing and mini-puzzles

Four were initially planned for the scenes: the home scene, a lake/pond, the forest scene, and a blue light biome. The limited time in the semester, as well as the difficulties of integrating the different aspects of scenes simultaneously being developed the scope of this was unreasonable, forcing the reduction from the planned four scenes down to just two, being the home/farm scene and the forest scene.

Regarding the extra game mechanics (i.e., fishing and mini puzzles), they were cut from the development process for the same reasons as the scene reduction state above.

For the version of the game ported over from Unity to the engine, several key changes were made:

1. Original plan: implement the Unity version of the game on the engine, modified to only implement the forest scene



*Figure 35 - Appendix 2: Initial Home Scene*

For this change, we pivoted from the idea of implementing the entire Unity prototype onto the non-euclidean engine to only implementing a modified forest scene. This change was brought about by the limited time in a semester, compounded with delays due to difficulty with the math for the Non-Euclidean shaders and mapping. As a result, the remaining time was not enough for a full implementation of the game, so we implemented a modified version of the forest scene because it was the best way to demonstrate the Non-Euclidean aspect of the game engine.

## APPENDIX 3 – OTHER CONSIDERATIONS

### A: Personas and Empathy Maps

#### Personas

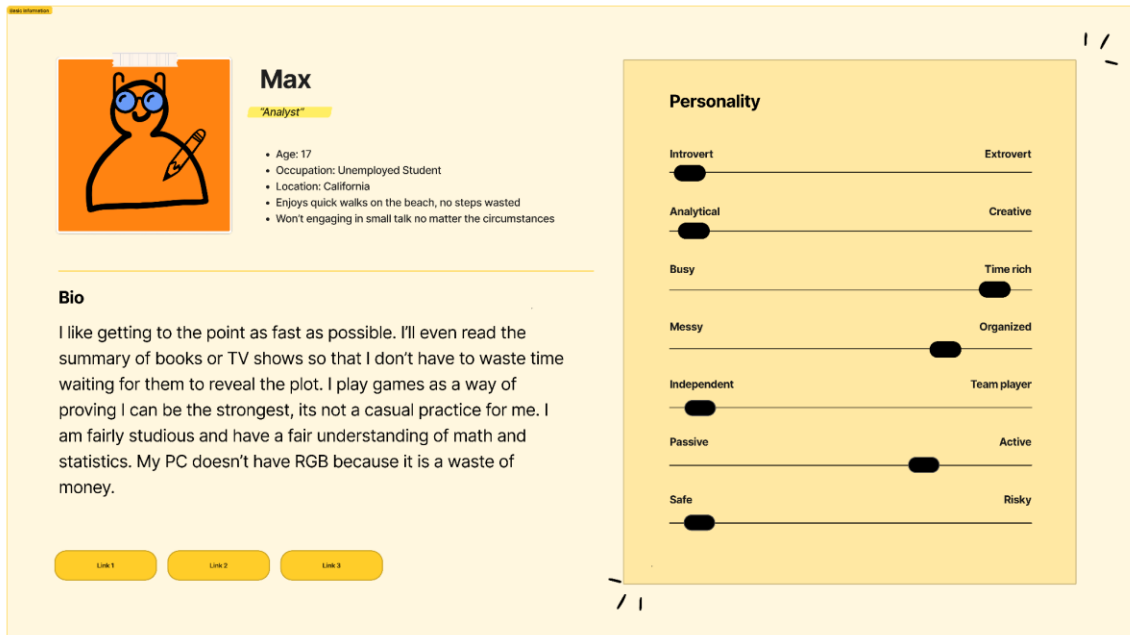


Figure 36 - Appendix 3: Max User Persona

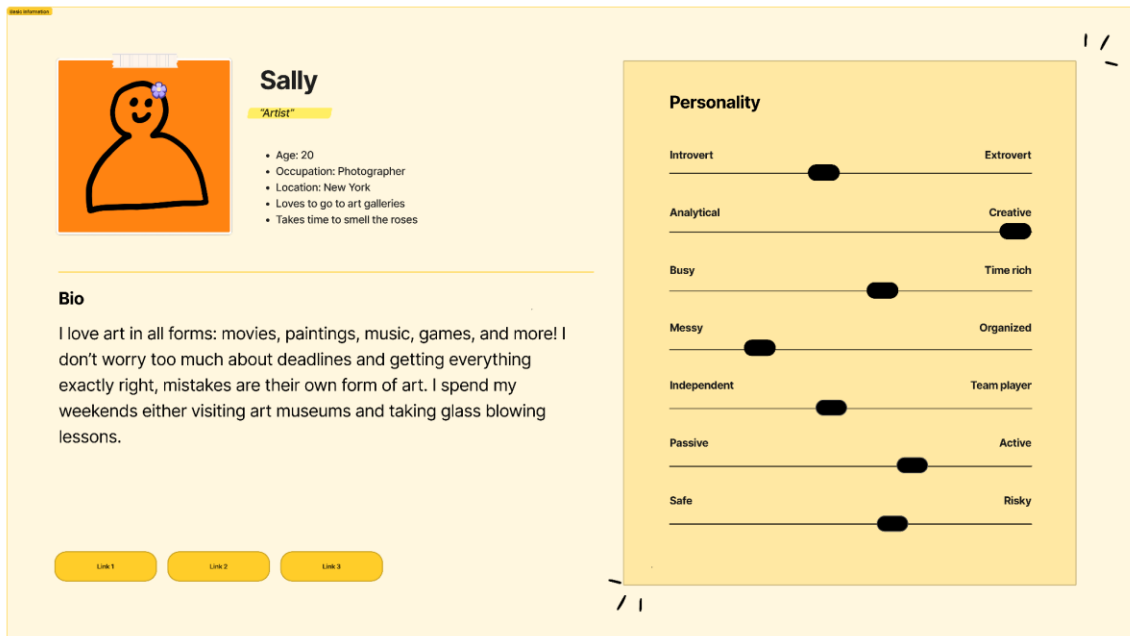


Figure 37 - Appendix 3: Sally User Persona

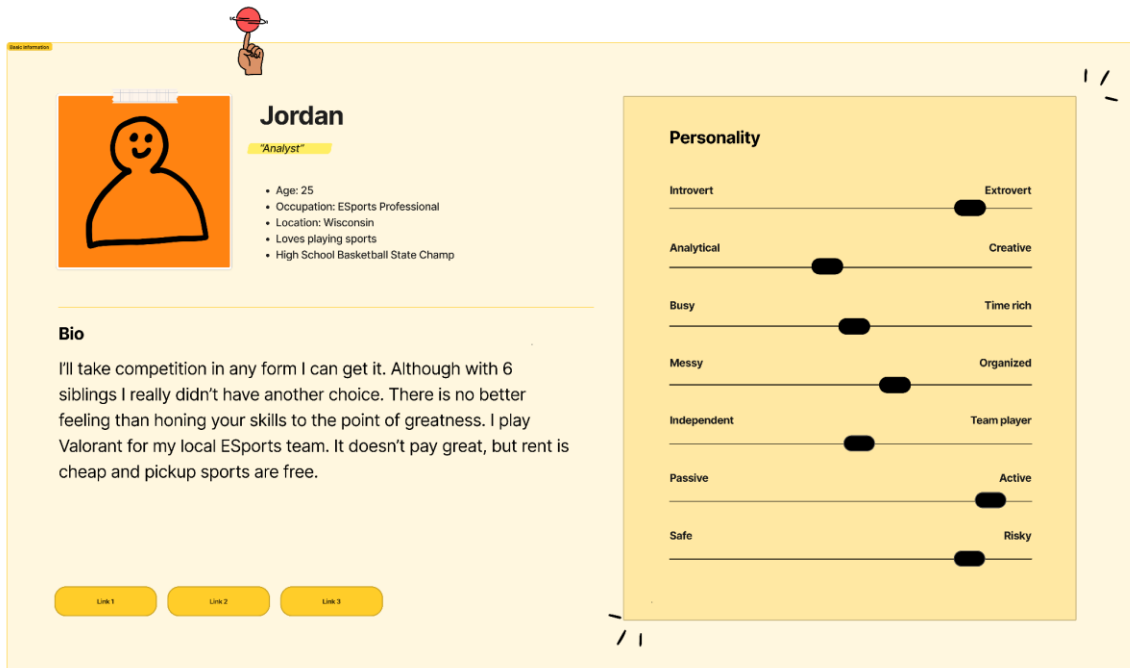


Figure 38 - Appendix 3: Jordan User Persona

Empathy Map:



Figure 39 - Appendix 3: Empathy Map

Playtest Document

# Lights Out Playtest

The game takes place on your grandfather's secluded farm, nestled deep in a rural countryside with minimal contact from the outside world. Upon arriving for a visit, the player discovers that their grandfather has mysteriously disappeared. With little to go on, the player begins to explore the area, growing and trading various seeds to acquire tools that may help uncover clues about their grandfather's whereabouts.

Main Objective: find your grandfather

---

## Controls

### General:

W / A / S / D – Move

SPACE – Interact with objects and characters (this includes planting and harvesting)

TAB – Open/close inventory

ESC – Open Settings/Pause Game/Controls Menu

### In the Forest:

Q – Place a lantern

E – Place a trap

### On the Farm:

E – Switch the type of seed you're planting

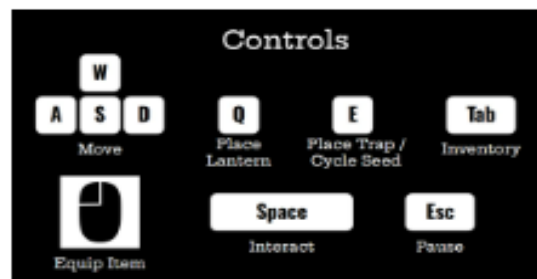


Figure 40 - Appendix 3: Playtest Document Page 1


## Feedback

We'd love your feedback! After playing, please take a moment to fill out our [short survey](#) and let us know about your experience. Your input is incredibly valuable and will help us improve the user experience in our game!


## Getting Started

Go to this [link](#) and download the zip file that matches your operating system (Linux or Windows). Unzip the file and open the folders till you see the file below and run it. It may try to stop you from running it. If so, select more info and hit run anyways.

Windows:

 Farming Prototype.exe	4/22/2025 4:43 PM	Application	651 KB
---	-------------------	-------------	--------

Linux:

 LightsOutLinux05_64	4/22/2025 4:50 PM	X86_64 File	15 KB
---	-------------------	-------------	-------

There is a web version of the game that can be found [here](#), but it can have a severely reduced frame rate and transition speed so we recommend downloading if possible.

## Read Before You Play

- Your plants grow while you are in the forest so go explore after planting
- Traps will capture enemies that walk over them
- Enemies will avoid light, so place a lantern to keep them at bay
- Traps and lanterns disappear after you leave the forest so use them sparingly
- If you die there is a chance your plants will disappear so be careful

Figure 41 - Appendix 3: Playtest Document Page 2

## APPENDIX 4 – CODE

Code Repo: <https://github.com/sdmay25-37>

### Non-Euclidean Engine Submodules

The current file structure of the project is shown in Figure 42. As mentioned in Appendix 1, the directory is divided into folders and subfolders to provide compartments for each individual submodule present in the project.

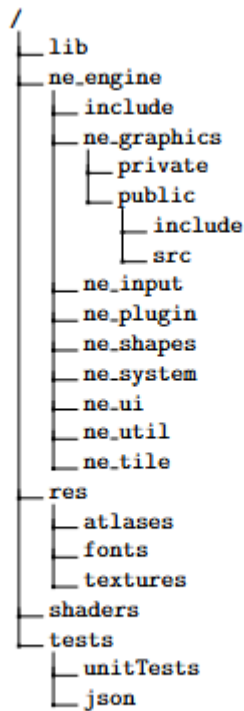


Figure 42 - Appendix 4: Current Repo Structure

Due to the sheer number of lines of code that we've written, snippets of code will not be included in this appendix, but a high-level overview of each folder and key submodule will be included in this section.

The following folders are used for assets, external libraries, and testing classes:

1. **lib/** contains external third-party libraries;
2. **res/** contains resources for the engine;
  - a. **atlases/** contains texture atlases for sprites;
  - b. **fonts/** contains fonts for displaying text;
  - c. **textures/** contains individual spritesheets for sprites and animations;
3. **shaders/** contains shader code used for the Non-Euclidean shaders;
4. **tests/** contains example classes using the engine for manual testing;
  - a. **unitTests/** contains unit tests written using the Catch2 framework.
  - b. **json/** contains tile mappings used for rendering

The Non-Euclidean engine has multiple submodules created from general abstractions needed to run a video game. Each module contains the following functionality:

1. **ne\_graphics/** - Graphics Submodule: The graphics submodule contains classes used for the render loop, managing shaders, managing textures, and storing images;

2. **ne\_input/** - Input Submodule: The input submodule contains managers for handling input to the GLFW window that the engine displays textures and shapes to, and loading stored keybindings from a JSON file.
3. **ne\_plugin /** Window submodule: The window submodule contains a wrapper class for the GLFWWindow for managing the window.
4. **ne\_shapes/** - Math Submodule: The math submodule contains code relating to hyperbolic rotations and the tessellations that are performed with the shaders;
5. **ne\_system/** - System Submodule: The system submodule contains a scheduling subsystem that optimizes the processing of subsystems;
6. **ne\_ui/** - User Interface Submodule: The UI submodule contains functionality for using a GUI to manually create and manage texture atlases, animations, files, and key mappings to allow for an intuitive development experience for the game design team;
7. **ne\_util/** - Utility Submodule: The util submodule contains various helper tools created for optimizations and general miscellaneous use.
8. **ne\_tile/** - Tilemap Submodule: the submodule contains the classes used to map tiles and figure out what sprites to render where

## External Libraries

### *General Management*

#### **ENTT**

ENTT is the Entity-Component-System (ECS) used for managing resources and custom components. The library manages the registry system, which is used for resource management and allocation. Through this ECS, custom components are registered for use and management, allowing for a programming style similar to Unity development [9].

### *Graphics*

#### **Glad**

Glad is the primary library used in conjunction with GLFW to expose the OpenGL graphics API for graphics programming. The library is not directly used in terms of API calls, but the library itself allows us to make custom shaders with OpenGL [10].

#### **GLFW**

GLFW is the wrapper library for Glad, allowing for window display, input capturing, etc. The GLFW library is the primary library used for rendering and window management, proving imperative for the project. The Input management and rendering subsystems are built on GLFW functionality (action callbacks and graphics buffers) [11].

#### **ImGui**

Additional library to create 2-dimensional Graphical User Interfaces.. Used for creating menus, displayable text, and interface for texture atlasing [12].



## Math

### Eigen

Eigen provides various mathematical functions in C++, often used instead of standard libraries. The library itself was primarily used in the prototyping phase, demonstrating proof-of-concept demonstrations of the hyperbolic geometry in a normal C++ source file as opposed to the final shader mathematics [13].

### Glm

glm provides fundamental data structures of multiple sizes, used in the graphics pipeline and storage of float or position information. The library provides matrices and vectors of size 2-4, which are used extensively in the shaders to provide rotations on the points in space. Additionally, the vectors are used for passing data from stage to stage in the graphics pipeline (vertex to fragment shader) [14].

## Loading/Writing Files

### JSON

Nlohmann's JSON library [15] is used for loading and storing keybinds from files, providing offline storage of the keybinds, used primarily for saving custom keybinds without hardcoding the values. Additionally, the library provides simple reading and writing to JSON formats, allowing for simple parsing and processing of JSON data [15].

### Stb\_image

stb\_image is a graphics library used for loading JPEG files. The library is utilized primarily for loading spritesheets to be used and managed for sprites, animations, and atlases.. The single include had to be modified due to issues with the header guard failing upon including in multiple files[16].

## APPENDIX 5 – TEAM CONTRACT

### Team Members

- Tasman Grinnell
- Joshua Deaton
- Lincoln Kness
- Ben Johnson
- Zach Rapoza
- Spencer Thiele
- Cory Roth

### Required Skill Sets for Your Project

- Basic Coding Skills
- Strong Math Background (geometry, calculus)
- Ability to work well with others
- Basic understanding of OpenGL

- Basic understanding of Unity
- Ability to be flexible with schedule
- Software Architecture Design
- Git
- Creative Skills
- Software Development Practices

### Skill Sets Covered by the Team

Skill	Covered By
Basic Coding Skills	Everyone
Strong Math Background (geometry, calculus)	Josh, Tasman, Ben
Ability to work well with others	Everyone
Basic understanding of OpenGL	Ben, Josh, Tasman
Basic understanding of Unity	Spencer, Lincoln
Ability to be flexible with schedule	Everyone
Software Architecture Design	Cory, Lincoln, Spencer, Ben
Git	Everyone
Creative Skills	Spencer, Zach, Ben, Lincoln
Software Development Practices	Cory, Tasman, Spencer, Ben, Lincoln, Zach

Table 12 - Team Skillset

### Project Management Style Adopted by the Team

This project adopted a hybrid of both waterfall and agile approaches. The overarching method consisted of a waterfall approach for the high-level structure of the timeline and task decomposition, but the specific task completion used an agile approach. This structure was chosen mainly because of the time constraint given. The timeframe for this project was only 2 semesters to produce the working final deliverables. Due to this constraint, a more rigid schedule is favored to ensure deadlines are met. This rigid schedule better aligns with a waterfall approach.

Another reason this approach was adopted was the dependency of tasks. Specific tasks depend on the previous task being completed, so a more agile approach cannot be taken. Things can only be iteratively improved upon when there is something to be improved upon. Once the group developed a semi-functional video game prototype, a more iterative approach was taken towards tasks and problems. This meant that there was a shift over the year from a heavy focus on waterfall to a heavier emphasis on the agile approach.

Finally, an agile approach to completing tasks was chosen because it made changes to the requirements easier. As this was a student-proposed project, the client was a student, which allowed for more frequent client feedback. It also allowed the project's direction to be changed

more seamlessly. This allows for more flexibility with the requirements and improved group decision-making.

### Individual Project Management Roles

Individual	Roles
Tasman Grinnell	Project Manager Render Engineer Advisor Interaction
Spencer Thiele	Game Designer Prototype Lead
Zachary Rapoza	Game Designer Art Lead
Lincoln Kness	Game Design Lead Web Master
Joshua Deaton	Render Engineer Math Lead
Benjamin Johnson	Render Engineer Lead System Engineer
Cory Roth	Note Writer Game Designer

*Table 13 - Team Member Roles*

### Team Contract

#### Team Members:

- 1) \_\_\_\_\_ Joshua Deaton \_\_\_\_\_
- 2) \_\_\_\_\_ Tasman Grinnell \_\_\_\_\_
- 3) \_\_\_\_\_ Benjamin Johnson \_\_\_\_\_
- 4) \_\_\_\_\_ Lincoln Kness \_\_\_\_\_
- 5) \_\_\_\_\_ Zachary Rapoza \_\_\_\_\_
- 6) \_\_\_\_\_ Cory Roth \_\_\_\_\_
- 7) \_\_\_\_\_ Spencer Thiele \_\_\_\_\_

#### Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
  - Face to Face weekly meeting on Wednesday from 1:30-3:00 for a larger team.
  - Smaller team meetings will vary weekly.
  - Meet as a large team in a library group room.
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., email, phone, app, face-to-face):

- The majority of communication updates, reminders, issues, and team meetings should be done through the Discord channel.
- Scheduling meetings with the advisor will be done through email with the team cc'd.

3. Decision-making policy (e.g., consensus, majority vote):

- Important decisions about the development of this project should be relative consensus.
- More minor issues can be decided with a majority vote.

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

- Every meeting, someone will take notes about what was discussed and make a list of issues of what to accomplish before the next meeting.
- Meeting minutes and notes will be posted in the shared Google Drive.

### *Participation Expectations*

1. Expected individual attendance, punctuality, and participation at all team meetings:

- Excluding the case where there are prior commitments (i.e., other classes/labs/etc), members should attend every meeting.
- If a member cannot attend a meeting, they must notify the team beforehand.
- Team members should be on time for meetings.
- Team members should actively participate in meetings and give their input to influence how this project develops.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

- As we break down what tasks must be done (in weekly meetings), we will assign tasks to team members. We will also assign a tentative deadline for when we expect the task to be completed.
- Team members are responsible for completing their tasks on time, notifying the team if such a deadline is not reasonable, and setting a new reasonable deadline.
- A general timeline is outlined in the project description of when vital milestones should be accomplished. It is the responsibility of the group to assign tasks to members to stay within the bounds of the timeline. There is also a general understanding that the timeline may change as the project progresses.

3. Expected level of communication with other team members:

- Team members should communicate with other team members when needed. As we are breaking into groups for the project, there should be a good line of communication in each group to ensure that each member is on the same page.
- A team member is responsible for communicating with other members if/when unexpected circumstances arise.
- Non-in-person communication should be over Discord
- Team members should respond to important Discord messages within 24 hours

4. Expected level of commitment to team decisions and tasks:

- Team members should commit to this project just as any other project-based class, understanding that most work will be done outside of scheduled class time and on their own time.
- Team members should be willing to go forth with decisions made by the team and should have announced their issues with any decisions when the decision was made.

### Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

Individual	Role	Responsibility
Tasman Grinnell	Project Manager	Manage the project In charge of communication with our advisor (Dr. Zambreno) Ensures that the team goals are being met Facilitates communication between teams Implement Features in Game Engine
Spencer Thiele	Game Design Lead	Responsible for leading the Game design team, including setting up meetings, managing scope, and ensuring the Game Design timeline is met Also responsible for making significant progress in Game Design, Prototyping, and Game Engine Development
Zachary Rapoza	Game Design Engineer	Responsible for flushing out sprites/art In charge of prototype testing Responsible for creating and testing mechanics for the game
Lincoln Kness	Game Design Engineer	Game Design, as well as keeping our website up to date Responsible for creating and testing mechanics for the game
Joshua Deaton	Rendering Engine Lead	Set up meetings with the rendering team Encourage progress on the rendering engine is being made Encourage collaboration on the rendering team Build a low-level implementation of the rendering engine

		Focus on getting a working Non-Euclidean model
Benjamin Johnson	System Engineer	Build a low-level implementation of the rendering engine Develop additional features to make a functional game engine
Cory Roth	Rendering Engine Engineer	Build a low-level implementation of the rendering engine. Develop additional features to make a functional game engine. Ensures assignments are completed on Canvas. Take notes at every meeting

*Table 14 - Leadership Roles*

2. Strategies for supporting and guiding the work of all team members:

- Small teams: Our strategies will follow a similar path to the agile development cycle for small teams. In the small teams, we will set goals for bi-weekly 'sprints.' We will also have a regular weekly check-up, with the ability to schedule smaller check-up meetings when necessary.
- Between teams, we will have a weekly meeting to discuss any significant issues that the teams face, along with syncing the process between the engine and design stages when necessary.

3. Strategies for recognizing the contributions of all team members:

- Since we will be working in smaller teams, the individual team leader will check to see what has been accomplished by the team, whether that is overcoming a blocker or just finishing a challenging section. The team leaders will then bring up the accomplishments at the big team meeting the week following the end of the small team meetings.

*Collaboration and Inclusion*

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

Member	Skills, Expertise, and Unique Perspective
Spencer	Avid interest in game development and already has prior experience in game design. He has experience in C#, Unity, and WebGL.
Josh	Experience with C/C++ and CMake. He is also decent at math and will help with the Non-Euclidean development.
Zach	Experience in C and some in C++, along with an interest in game design.

Ben	Previous experience with OpenGL and game engine rendering. Currently taking Computer Graphics class.
Lincoln	Experience in C/C++ as well as JavaScript, CSS, and HTML. Is interested in game development and has experience with Unity.
Tasman	Experience in C/C++. Has an uncanny skill of understanding complex technical details.
Cory	Experience in and enjoyment of debugging. Has experience with C/C++.

*Table 15 - Member Skills*

2. Strategies for encouraging and supporting contributions and ideas from all team members:

- Reaching out to a team member and asking if they are stuck on understanding or in writing a code block
- When giving opinions on an idea or contribution, team members should point out a positive feature when providing constructive feedback.
- Team members should understand that everyone has a unique perspective and that their point of view is still valid even if they disagree with their idea.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment obstructs their opportunity or ability to contribute?)

- If team members believe the environment obstructs their progress, they should bring it up in the weekly small team meetings or in their respective Discord channels. They should also arrange a time to talk with the team about how we should propose a solution.

### *Goal-Setting, Planning, and Execution*

1. Team goals for this semester:

- To develop a design and implementation plan for a Non-Euclidean game engine and a game to run on top of that engine.
- To start developing the early stages of the game and engine.
- Promote a positive group environment where each individual feels comfortable contributing to the group.

2. Strategies for planning and assigning individual and team work:

- We will break down the team of 7 into two separate teams, focusing primarily on the game development and Non-Euclidean engine aspects.
- Within those specific teams, work will be divided and assigned as the team decides on what needs to be done.

3. Strategies for keeping on task:

- Create a clear and structured plan
- Follow the plan

- Communicate if there are issues early and often

### *Consequences for Not Adhering to Team Contract*

1. How will you handle infractions of any of the obligations of this team contract?

- If a team member cannot meet one of their obligations, the other team members will try to talk with them and to figure out if we can assign different tasks, work together to accomplish the task or other possible solutions.

2. What will your team do if the infractions continue?

- If this issue arises, we will bring it to the attention of our advisor on how to continue, but we believe this will be fine in our project.

\*\*\*\*\*

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) \_\_Tasman Grinnell\_\_\_\_\_ DATE \_9/17\_\_

2) \_\_Cory Roth \_\_\_\_\_ DATE \_9/17\_\_

3) \_\_Lincoln Kness \_\_\_\_\_ DATE \_9/17\_\_

4) \_\_Joshua Deaton\_\_\_\_\_ DATE \_9/17\_\_

5) \_\_Zachary Razpoa\_\_\_\_\_ DATE \_9/17\_\_

6) \_\_Benjamin Johnson\_\_\_\_\_ DATE \_9/17\_\_

7) \_\_Spencer Thiele\_\_\_\_\_ DATE \_9/17\_\_