Lights Out

Design Document

Team 37

Client: Josh Deaton

Adviser: Dr. Joseph Zambreno

Team Members:

Tasman Grinnell	Project Manager
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 Render Engineer

Team Email: sdmay25-37@iastate.edu

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

Revised: 12/07/2024 Version 1.0

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 engine for true 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 render engine must be custom.
- Engine must render Non-Euclidean geometry.
- Standard games built with this engine must render at a minimum of 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 rendering engine, which supports the requirements specified. Individual submodules will support operation by managing memory, handling inputs, and drawing to the screen. The engine will operate on a loop, handling inputs of Euclidean data, transform it into the Non-Euclidean space, and then project the data back to the user on a display screen. This engine will be developed in C++ with OpenGL and run on a personal laptop. The game that is designed to show this engine is called *Lights Out*. It is a farming simulator that takes on a horror aspect using lighting mechanics.

The current progress of this project:

- The game has been designed and fleshed out regarding the story and world environment.
- The game's main functionalities have been prototyped.
- The rendering engine has a framework structure to render sprites and assets.
- The rendering engine can handle inputs from the computer.

Given the progress made, the current solution still addresses the needs of our users. The primary user need is an enjoyable experience, and given the development of our game, that need is met. Given the technical requirements of rendering it Non-Euclidean, the team is aware that focus must be put on making the gameplay smooth. Progress toward the solution of the rendering engine, creating an engine that runs optimally is still a possible issue.
The plans for this project include starting the implementation of features in the render engine that the game design needs. Another task will be completing the library that will be used to render the Non-Euclidean math. In terms of game design, the future plans combine prototypes to build a working demo in unity. This way, the design can be tested from a user needs and constraints perspective. Progress can only be made to test the technical requirements once the render engine is complete. Once both are complete, the main task will be the game design and render engine integration.

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

Table of Figures

Table of Tables

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 games that use Non-Euclidean geometry 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. The goal of this project is to create an engaging and interesting game with a custom rendering engine to support the actual implementation of Non-Euclidean worlds.

Some of the most prominent issues in creating true Non-Euclidean games involve performance due to the number of 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 who will use 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 8.3.A.1]. 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 8.3.A.2]. 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 8.3.A.3] 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 on combating the 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 rendering 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.

Rendering Engine

Type of Requirement	**Requirement**
Functional	Renders Non-Euclidean geometry
Functional	Renders vectors of data into a laptop
Functional	Run at 30 fps (**constraint**)
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 - Rendering Engine Requirements

Game Design

Type of Requirement	**Requirement**
Functional	Easy to understand, yet interesting game mechanic
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	East to use controls
User experiential	Easy-to-follow story
Economic/Market	Easy to monitor and control 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	Some 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 rendering 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-bone project but a fully fleshed-out idea. The aesthetic and user experiential requirements will set this project apart from other projects 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 failure 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 rendering 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 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 relevant. 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 relevant 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 develop 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 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 interface were researched.

Some modifications to the project design will be made to incorporate these standards. Given the standards looked at, the design documents about low-level features might mostly stay the same. However, the design details of the final deliverable and how the project will be expressed to the users will change. More detail and attention will be given to the vocabulary used in documentation and gameplay for the video game. Once it works, a change is needed to design the video game's management. The framework that needs to be put in place needs to be created. It must make sense with the other requirements for this project. The design of how testing is done may also change to the standard for quality assurance.

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. The timeframe for this 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 that a shift can happen over the year from a heavy focus on waterfall to an increasing focus 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.

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

3.2 Task Decomposition

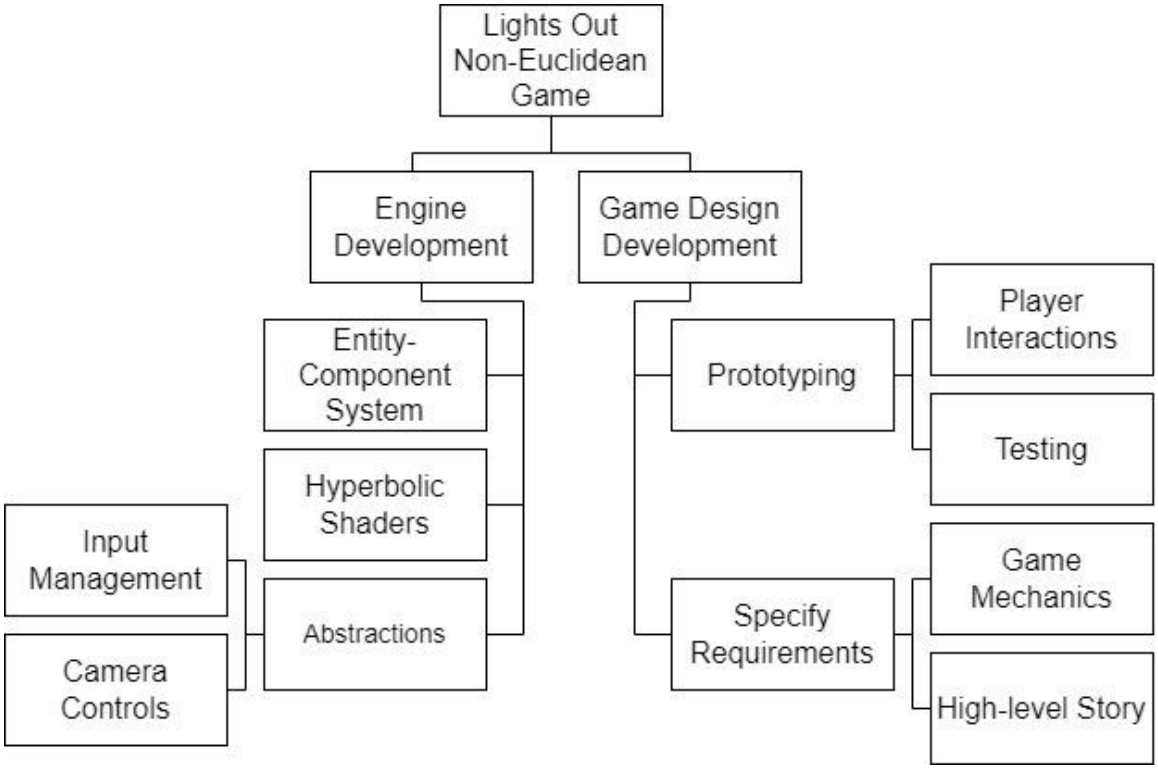

Figure 1 - Task Decomposition

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

The game design team will be performing the right branch of the task decomposition tree, pictured above, while the engine team will be operating 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 the creation of 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 gameplay loop.

Rendering Engine Team Milestones

- Math Determination
    - 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 link:
        - Sprites
        - Entities
        - Lighting
        - Collision
- Math Implementation
    - Ability to render the above in a Non-Euclidean manner.
- Running Video Game on Engine
    - Ability to run all of the video game scenes created by the game design team

3.4 Project Timeline/Schedule

Gantt Charts

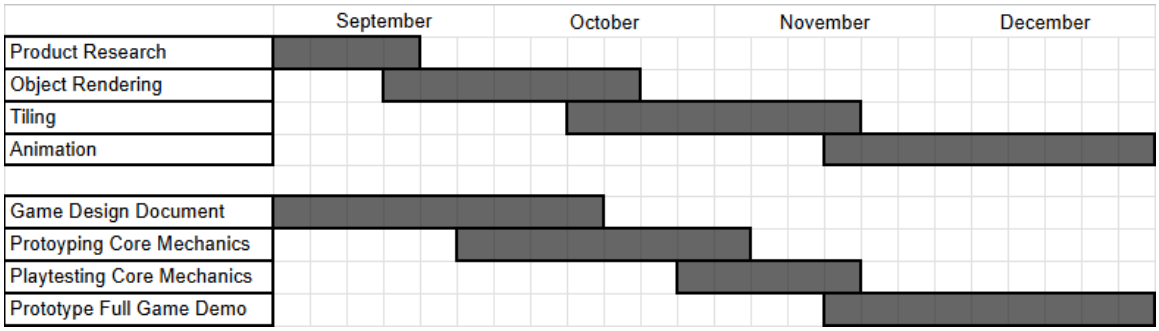

Figure 4 – High Level Gantt Chart

GANTT CHART Game Design

PROJECT TITLE	Lights Out
PROJECT MANAGER	Tasman
COMPANY NAME	Sdmay25-37
DATE	10/30/24

WBS NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION
1	Product Research			
1.1	Game Type Research	9/4/24	9/12/24	8
1.1.1	Game Resources Reseach	9/6/24	9/14/24	8
1.2	Idea Generation	9/16/24	9/20/24	4
1.3	Game Selection	9/19/24	9/30/24	11
2	Design Document			
2.1	Lore Generation	10/1/24	10/6/24	5
2.2	World Enviroment	10/4/24	10/10/24	6
2.3	NPCs	10/7/24	10/12/24	5
3	Protoying Core Mechanics			
3.1	Player Movement	10/14/24	10/21/24	7
3.2	Monster	10/14/24	10/21/24	7
3.3	Farming Mechanic	10/16/24	10/23/24	7
3.4	Lighting	10/16/24	10/23/24	7
3.5	Storage	10/18/24	10/25/24	7
4	Integrating Prototypes			
4.1	Single Scene Creation	10/21/24	10/31/24	10
4.2	Multi Scene Creation	10/28/24	11/8/24	10
4.3	Basic User Testing	11/4/24	11/15/24	11
5	Demo Creation			
5.1	Creation of Working Demo	11/18/24	12/2/24	14
4.1	Testing Working Demo	12/2/24	12/20/24	18

Figure 3 - Game Design Gantt Chart

GANTT CHART Rendering Engine

PROJECT TITLE	Lights Out
PROJECT MANAGER	Tasman
COMPANY NAME	Sdmay25-37
DATE	10/30/24

WBS NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION
1	Product Research			
1.1	Engine Research	9/4/24	9/12/24	8
1.1.1	Math Research	9/6/24	9/24/24	18
1.2	Engine Selection	9/20/24	10/4/24	14
1.3	Math Selection	9/23/24	10/4/24	11
2	Basic Rendering			
2.1	Creating 2-d shapes	10/1/24	10/6/24	5
2.2	Basic Ideas	10/4/24	10/10/24	6
2.3	Basic Math Concepts	10/7/24	10/12/24	5
3	Prototyping			
3.1	Sprites	10/14/24	10/21/24	7
3.2	Klein Model	10/14/24	10/21/24	7
3.3	Camera Movement	10/16/24	10/23/24	7
3.4	Lighting	10/16/24	10/23/24	7
3.5	Assets	10/16/24	10/23/24	7
3.6	Input	10/18/24	10/25/24	7
4	Complex Issues			
4.1	Lighting	10/21/24	10/31/24	10
4.2	Collision System	10/28/24	11/8/24	10
4.3	Math Implementation	11/4/24	11/15/24	11
5	Demo Creation			
5.1	Creation of Working Demo	11/18/24	12/2/24	14
4.1	Testing Working Demo	12/2/24	12/20/24	18

Figure 2 - Render Engine Gantt Chart

By the end of this semester, The two deliverables will be a rendering demo and a video game demo. These deliverables will be completed by the end of the semester due to the build-up work that must be done. By the end of the year, our final deliverables will be a working rendering engine that supports a video Non-Euclidean game. Because the video game deliverable depends on the working rendering engine deliverable, the rendering engine must be completed before the video game can be completed.

These Gantt charts show the proposed schedule for the project this semester. It is divided into two main groups: Game Design and Rendering 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 rendering engine team. Both teams start with doing project research. The rendering engine will spend more time on the Non-Euclidean math in this part of the project due to their heavier reliance on it. The game design team will primarily focus on researching what makes a video game suitable to create 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 rendering engine will operate.

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 start-of-the-art tools are.
Idea Generation	Lack of Innovation Game Choice	0.6	See Mitigation
Game Selection	Choosing a game with too big of 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 rendering 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

- **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 being 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 rather 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 still arise.

  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.

Rendering 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 plentiful tutorials, so it should not be a major issue.
Basic Ideas	Don't properly learn to render, so features take longer	0.3	There are plentiful tutorials, so it should not be a major 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 it is possible that the chosen idea is 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 it comes to testing/verifying correctness.

Table 5 - Render Engine Risks

Rendering Engine Risk Mitigations

- **Math**

  The main risk for the rendering engine is being able to render Non-Euclidean math. None of the team members are math majors, along with this 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 adequately understand the math.

  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, risk is mitigated that incorrect information was learned. A list of resources that can be used is also created in case advice is needed on how to better this understanding.

3.6 Personnel Effort Requirements

Game Design Hours

Task	**Estimated Person Hours**	**Reason**
Product Research	100	Want to spend 2-2.5 weeks coming up with a good idea.
Game Type Research	20	Need to determine what users like/dislike. Each member puts in 5 hours so everyone has an idea.
Game Resources Research	15	This can be done relatively quickly, but members should know what resources exist.

Idea Generation	40	There should be four people taking ~1 week time to create a good initial plan.
Game Selection	25	~6 hours per member to flesh out ideas and ensure solid ideas exist.
Design Document	85	The game needs to be interesting. Time needs to be spent making it so.
Lore Generation	25	Lore is important to keep the players invested.
World Environment	30	Exploration is a significant part of game design, which needs an exciting world.
NPCs	30	It is essential to why certain games are loved/hated.
Prototyping Core Mechanics		Each core mechanic is being given to one person with 1.5 weeks to implement and test.
Player Movement	15	1.5-2 weeks' worth of work for one person.
Monster	15	1.5-2 weeks' worth of work for one person.
Farming Mechanic	15	1.5-2 weeks' worth of work for one person.
Lighting	15	1.5-2 weeks' worth of work for one person.
Collision	15	1.5-2 weeks' worth of work for one person.
Storage	15	1.5-2 weeks' worth of work for one person.
Integrating Prototypes	200	Very important, the team should spend a month making a solid game.
Single Scene Creation	40	~ 1 week to integrate core mechanics.
Multi Scene Creation	80	~ 2 weeks to create more scenes and add more mechanics. Allows for documentation and preparation for the second semester.
Basic User Testing	80	~ 2 Weeks. There will be bugs, time is needed to test/ change implementation based on feedback.
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.
Creation of Working Demo	80	~2 weeks to get a polished demo.
Testing Working Demo	70	The goal is to make a smooth game.

Table 6 - Game Design Person Hours

Rendering Engine Hours

Task	Estimated Person Hours	Reason
Product Research	70	~ 2 weeks to get initial research done.
Engine Research	20	Want to do it right the first time.
Math Research	30	Time is needed to be spent on understanding the complexities.
Engine Selection	5	Each member has one one-hour meeting.
Math Selection	5	Each member has one one-hour meeting.
Basic Rendering	80	Coupled with research = ~ 1 month time
Creating 2-d shapes	25	It is needed so baseline information is there.
Basic Ideas	30	Time to explore OpenGL and get used to the software.
Basic Math Concepts	25	The math is complex..
Prototyping		Giving members ~1 week to complete each prototype.
Sprites	10	1 week.
Klein Model	40	One month because, most important, this needs to be working exceptionally well.
Camera Movement	10	1 week.
Lighting	10	1 week.
Assets	10	1 week.
Input	10	1 week.
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.
Collision System	40	~ 2 weeks with two people.
Math Implementation	80	~ everyone will spend 10-20 hours rendering complex math.
Demo Creation	200	Overestimate. Assuming previous tasks will take more time. Want to create a solid product.

		~ 1 month and some.
Creation of Working Demo	120	
Testing Working Demo	80	

Table 7 - Render Engine Person Hours

3.7 Other Resource Requirements

In terms of game development, additional resources will be needed, as discovered during the prototyping phase of the game design and development. The additional resources needed are:

- Sprites, Images

  For the game development itself, using sprites is 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 are art students, acquiring sprites and images created by professionals would be preferable and result in an overall 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. Writing custom code to perform these tasks will be impossible in addition to the general engine development required in the course's timeline. Therefore, using established libraries is 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 the development, viewing, and creating of the 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 merge conflicts with other version control systems. The Game Design team has used the free version of UVC that allows up to 5GB of storage space. However, funding may be required for the pro version of UVC as the final game may exceed this 5GB limit.

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

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 designs 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 main focus for creating the rendering 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 rendering 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 for visualizing models, the API developed for the project 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	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: ● Mental Well-Being and Cognitive Development ● Educational Impact	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	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: ● Inclusivity and Accessibility ● Respect for Community Practices	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. 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.
Environmental	This project aims to encourage players to plant and grow flowers and take care of the environment around them. Key aspects include the following:	Players gain positive connotations associated with growing crops and flowers by having users grow and maintain crops. The connotation

	• Eco Friendly • Biodiversity	may inspire users to grow plants and flowers and create gardens. Some potential benefits are increased biodiversity and providing bees with more pollen sources.
Economic	This project has various economic impacts at the individual, organizational, and broader community levels. Key aspects include the following: • Affordability for Consumers • Broader Market Impact	To ensure accessibility, the game and rendering engine must remain affordable for a broad audience, avoiding high price points that could alienate potential users. By pioneering a genuinely Non-Euclidean rendering engine, the project could establish a niche market, creating opportunities for partnerships, licensing, and expansion into educational and research sectors.

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 5 - Poincaré model example [5]

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 very hard 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 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 rendering 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 high mathematical understanding and many hours to read and understand the published articles and papers. Furthermore, the level of mathematical knowledge 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, a significant amount of 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 rendering 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 created 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 need 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). Additionally, mathematical models were evaluated similarly, primarily with the accessibility and relevance to the high-level game design. Finally, the high level game design was determined based on team interest and providing an entertaining idea for the game design.

Key Decisions

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

- Chose to implement the rendering engine in C++ with OpenGL graphics
    - Why?
        - 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 able to 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 base 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 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 for why these games were good and how these games could be improved upon were listed. From this list, two main artifacts were derived: 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 any 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 5 is the best and 1 is the worst of the ideas. Below is the matrix that was created.

Game Ideas												
	Non-Euclidean Utilization	Other Technical Requiremnts	Effort	User Needs Appeal	Exceeds State-of-the-art	Replayability	Fun Factor	Developer Appeal	Story	Orginiality		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 6 - 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* and 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 possibility 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 a 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 of a game would be given the idea.

  Even though *Euclid-Sho*t 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 reachable 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 most minor 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 possibility 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 very 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.

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 that 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.

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

4.3.2 Detailed Design and Visual(s)

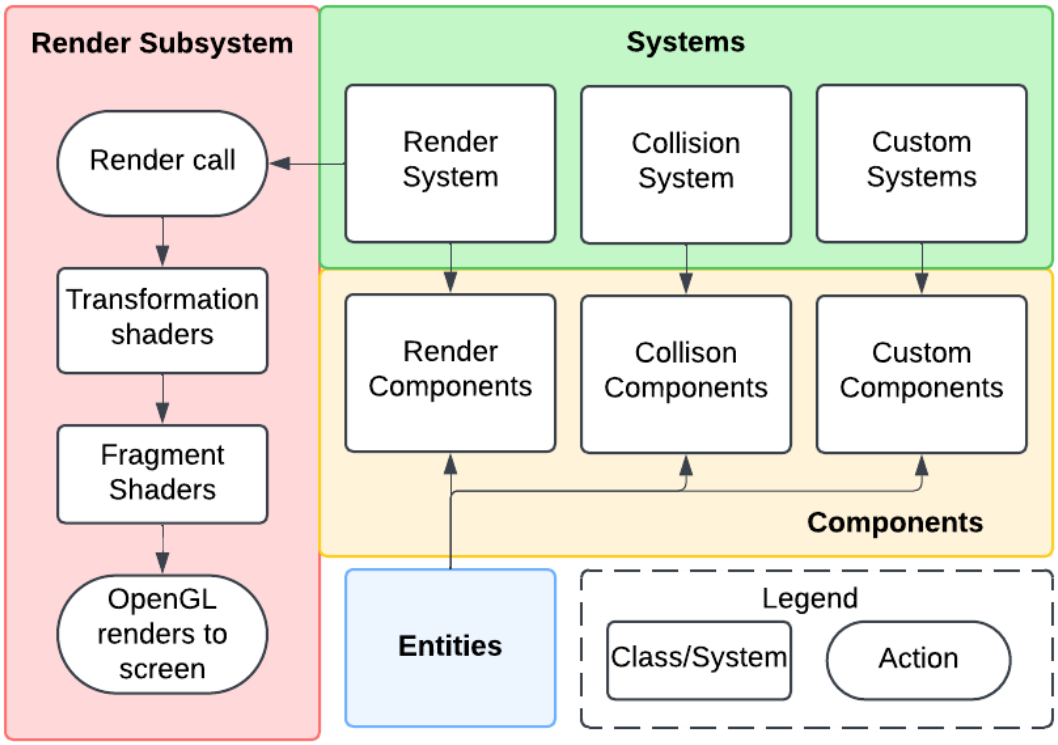

Figure 7 - Detailed Design and Visuals

The Rendering Engine

The rendering engine will process inputs from the game to effectively project and respond to actions performed in the game. Shaders will perform calculations to convert the Non-Euclidean world to Euclidean coordinates to be rendered to the screen. The Entity-Component-System will be responsible for managing resources while the primary rendering loop will be drawing 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.

Game Engine

The game engine is responsible for taking in inputs from the laptop and tasking the appropriate system on what to do. It will make calls to the Entity-Component-System, the rendering engine, or both. It will receive data from the rendering engine with the appropriate updates to the frame. This is the primary interface between the user and the rendering engine. The game engine will also be responsible for handling events that happen in the game. A significant feature that this system will have to manage is collision detection, which is necessary for many areas and entities of the world to be interactable.

Laptop

The laptop is the primary I/O device for the user and the system that the Game Engine, ECS, and Rendering Engine will run on. A laptop will consist of a keyboard that will be used as the input the user will make. 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 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 are intentional to force optimizations to increase the number of playable platforms, leading to increased accessibility.

4.3.3 Functionality

System Loop

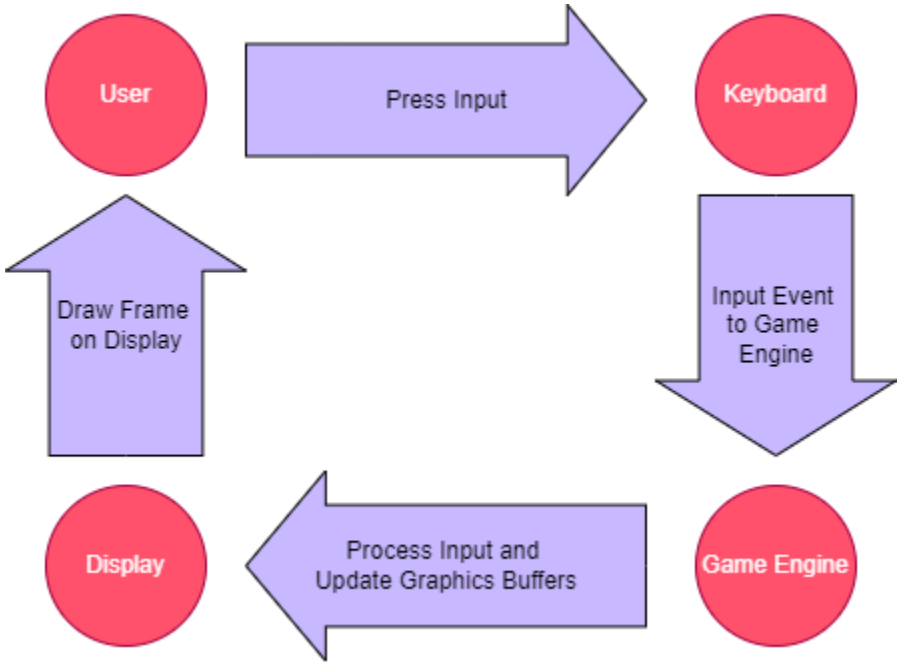

Figure 8 – User-System Feedback Loop

The user will interact with the proposed solution using a computer and some input device (keyboard or controller). When players play the game, they interact with elements such as moving the character, selecting inputs, or triggering actions. 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 render 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 rendering 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 will satisfy the requirements and meet the user's needs. Additional features will need to be implemented to confirm this, but based on the progress made by each team, these requirements are feasible. 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.

The immediate plan for developing solutions to address those concerns is to meet with our advisor to get guidance on how valid these concerns are. Scope readjustment will be discussed to determine the feasibility of a project that seems completable within the time constraints. Group meetings will be scheduled to determine if any changes to the design are needed. The major questions to our advisors are clarifying if he believes, given our progress, that the constraints can be met and asking internally if the design will meet the user needs.

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

Some tradeoffs are being made in determining the technology being used. The decision was made to use OpenGL as the graphics language, which allows cross-platform rendering – an important 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, but 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 rendering 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 limitation of resources. A different solution would be to make this game only run on computers with high-quality hardware parts, but the downside to this solution is that it may be harder to test those specs as well as decreased accessibility due to financial limitations.

4.5 Design Analysis

Here is a list of tasks that have been completed:

Game Design

- Product Research
- Idea Generation
- Game Selection
- NPC creation
- Biome creation
- Prototypes created
    - Player Movement
    - Lighting
    - Monsters
    - Farming
- Single Scene Created

Render Engine

- Product Research
- Math Research
- Basic Rendering
    - 2-d Shapes
    - Basic Ideas
    - Entity component system
- Prototypes Completed
    - Basic Sprite

    - Basic Asset
    - Input Handlers

Other things that have been built not specified above:

- Started lore document for the game
- Initial story plans for the game
- Rendering engine template/framework
- Started library for math render call

It is impossible to tell whether the proposed design works or not. Current progress is in the middle of developing the design. It can only be determined once the design is completed to give a valid answer to this question. The proposed solution has good design concepts for the game and a solid foundation for the rendering engine. The future plans for this project include starting the implementation of features in the render engine that the game design needs. Another task will be completing the library that will be used to render the Non-Euclidean math. In terms of game design, the future plans combine prototypes to build a working demo in unity. This way, the design can be tested from a user needs and constraints perspective. Progress can only be made to test the technical requirements once the render engine is complete. Once these are both complete, the main task will be integration. From the given work, the most significant implication for feasibility is integration. Completing a rendering engine within the time frame is feasible. It is also doable to complete the game within the timeframe, but integration may take longer than expected. That is because integrating Unity code with a custom engine is complex and many game design features will have to be reimplemented due to the difference in API functionality.

5 Testing

In the current state of this project, there is no system 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; unit testing relates to the implemented functionality. Interface testing is done by testing how method calls work with each other, and integration testing relates to testing the rendering of all the currently implemented functionality.

Even though there is no system in place, there is a plan for system testing. That section will cover how this group will test the game's integration on the render engine. This section will require more time and tests than the current design. Similarly, there is a brief plan for regression and acceptance testing. However, as the main focus is creating a working system, this plan is short. Once a working system exists, the focus will be shifted to developing a better regression testing setup as more and more functionalities will be implemented.

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 will be 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 is how the gameplay feels, which is very subjective. Therefore, manual testing is required for polishing gameplay and gathering opinions on the gameplay loop and mechanics. The edit mode of UTF will be used to ensure that objects and assets have the correct properties and settings. The play mode will be crucial for validating the correctness of functionality.

Manual tests will also give basic gameplay mechanics. Each mechanic that will be implemented will be separately created in its own space in unity. It will be made sure that the mechanic works separately through manual testing and playing through to make sure that it meets the requirements of the mechanic. This ensures that before integrating mechanics together, each mechanic works as intended.

The Engine will be tested using Google Test (GTest) and Google Mock (gMock) for unit tests. The game engine is much more straightforward to test due to the functionality and demanding performance requirements. Correctness is incredibly important, so using vetted testing suites for C++ to ensure correct values are output for each functional unit is ideal. GTest and gMock 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 involve the Game Engine and the Game Design portions of the project. The game will not connect to a server, meaning many interfaces will involve components internal to the project. For example, the primary gameplay loop will have to interface with Input Managers and Shaders, and the correctness of communication will have to be ensured through these tests.

In terms of the finalized product, the primary testing methods will be playtesting as well as gMock and GTest for the interface between components. The highly interconnected nature of the gameplay and the classes require close integration and efficient communication between objects and methods. Due to the custom nature of the engine, gMock and GTest will be used as the testing between classes with gMock being quintessential to mocking functionality from other classes.

5.3 Integration Testing

The critical integration paths within the design are in the game engine with response time in latency to ensure smooth gameplay. Due to the correctness ensured by unit tests, integration tests must 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 will need to be tested to ensure new additions are working as intended. Furthermore, monitoring performance will be critical to identify early in the development lifecycle.

GTest 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 focus 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 focuses 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., do certain mechanics need to be smoother or do they 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 properly, but 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, but given that the game is 2-dimensional, the likelihood of graphics card bottlenecks is very low, even with a large number of mathematical conversions.

5.5 Regression Testing

Regression testing will occur 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 as well as meeting performance and entertainment requirements. Additionally, playtesting will be able to 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 important piece to consider is performance requirements within the new updates. Profiling will continue to be important 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 will occur at this stage, with proper documentation being required at this stage. Furthermore, each performance target will be reevaluated at this stage, primarily meeting high-level framerate targets and memory requirements.

Written code that is examined at this point will be 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 the stage to ensure polished animations and responsive actions are present in the update.

5.7 Results

Due to the lack of a complete project, utilization of the above testing plans is sub-par. It is essential to have a working demo for all forms of testing besides the unit, as well as to have a comprehensive test set that provides an accurate view of the project. Many of the testing plans outlined above are unable to be completed until many of the components are completed. A heavier focus on testing will be placed in the optimization phase of development.

For example, using profiling for the tests will be able to determine the bottlenecks in the programs where there may be room for improvement and optimizations. Improving systems in this method is difficult when there is an incomplete game and engine, so multiple phases of testing and iterations upon the development will be in order.

Many of the tests currently in progress are primarily in regards to correctness, meaning that system, integration, and unit tests will be imperative for development to ensure edge cases are caught and fixed, as well as generally ensuring proper functionality. In the future, testing for subjective gameplay experience and profiling to identify performance bottlenecks will be much more imperative closer to a finalized product.

6 Implementation

Currently, 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. The engine uses a general loop to draw and manage resources for the main loop.

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 stages 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. If these mechanics can be implemented, a very base-level working demonstration can be achieved. 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 each other was 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 just combines the different parts and modifies certain aspects so that they work together seamlessly. The important part of this stage was making sure that when integration was happening, sprites and entities were consistent and that there were not two different ways of tiling used, for example. This stage also focuses on unifying the team's code standards.

Scene Development

In the third stage, the current stage of development, work is being done to create scenes. Two scenes are currently being 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 currently used in these scenes.

Figure 9 - Home Scene

The home scene is where the player will start off 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 at this time. 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.

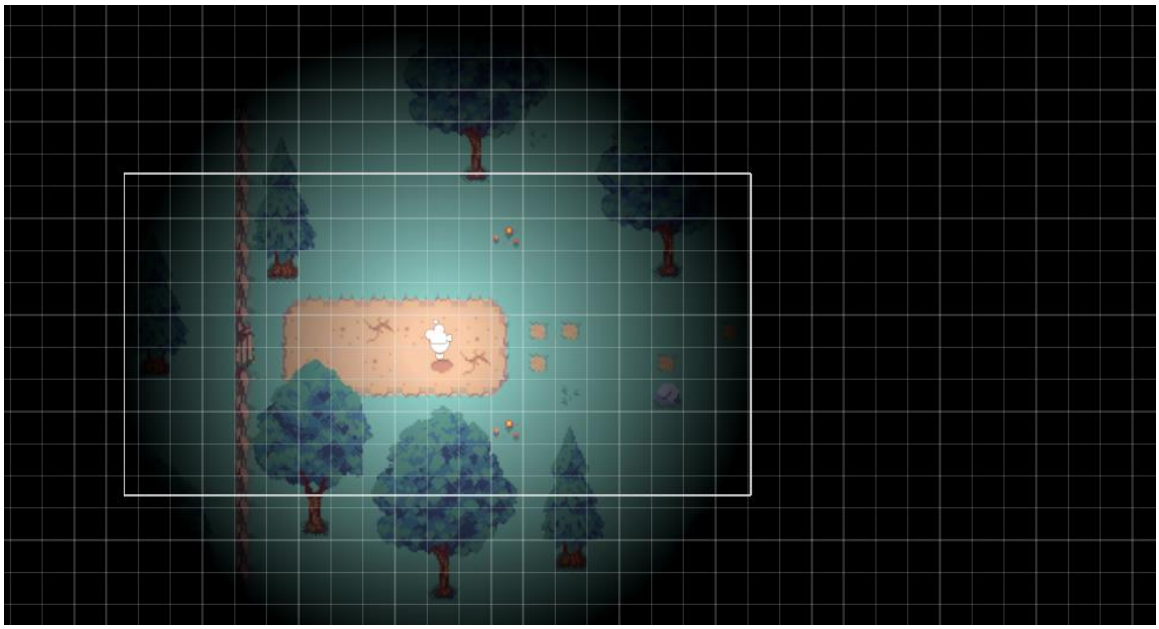

Figure 10 - Forest Scene

The forest scene, picture above, introduces the idea of the player needing to have some sort of a 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 will need to watch out for monsters that will avoid lanterns, but not the immediate field of vision light displayed currently.

There are 2 mechanics that this team still wants to work on before integrating with the rendering engine; NPCs (non-player characters) and traps. There are plans for implementing these mechanics and prototyping, however it has not been gotten to due to the current timeline. There are also auxiliary mechanics that are planned to be implemented once the game runs on the custom render engine, but are not vital to the game so are not currently in the queue.

6.2 Engine Implementations

The game/rendering engine is currently in a templatized state, with the main functionality setting up libraries and the windows for the rendering to occur. Sprite rendering, key mappings, and buffer management are currently implemented in the main rendering loop. Abstractions are currently in progress, primarily related to the camera controls and input management classes.

Figure 11 – 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.

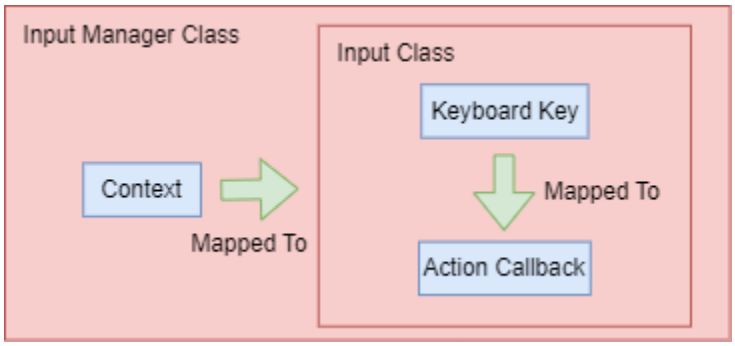

Figure 12 - Input Management Mapping Scheme

Input management primarily relies on a mapping given in Figure 12 as well as 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. Additionally, a high level input manager will handle the mappings from contexts (different settings require different functionality) to different input objects, which hold the relevant associations between keys and the functions.

7 Ethics and Professional Responsibility

The overarching team philosophy is utilitarianism. Our group aims to do good work in order to increase the overall happiness of our users. To do this, a solution that users will enjoy, use, and come back to is needed. Over the course of this project, this group aims to act professionally and respectively 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 Ethic 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 to be smaller than that 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 to 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 honestly communicate our progress within the team.
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 area of responsibility in terms of 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 such 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 in order 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 enjoyment to any users around the world.

Table 10 - Areas of Professional Responsibility

One Area Team is Performing Well: **Financial Responsibility**

The team is performing this area well, as seen with sticking to a low cost budget. Many software projects often have bloated budgets and set unrealistic expectations for personnel hours. The bloated budgets often 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 the management of game design files will be more difficult, with the benefit of reduced cost. Another way this team is reducing the budget is by using open-source software and resources. The rendering 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 upholds the ethic because the use of low-cost technology and resources allows for the project's budget to be well maintained.

One Area Team Needs to Improve: **Social Responsibility**

An area of responsibility this group can focus more on is social responsibility. In the current design of this project, there is a low impact on improving the social environment. This is due to the fact that the purpose of this project is for personal enjoyment. This project aims to bolster an individual's enjoyment, but not a society or community. Going forward, this will be changed. There will be an added effort towards making the rendering engine usual for the academic and research community. The render engine being created will model a game in Non-Euclidean space and create and create an API to make those calls. It would be reasonable to ensure that the interface being created will allow for mathematics and other research to visual 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 aims 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 ensures the game delivers positive, meaningful impacts to players. To achieve this, focus will be placed on balancing entertainment with learning and conduct user testing to refine how the game benefits users cognitively.

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

The project will most likely have 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 - beneficence by creating a quality open source engine that companies would like to reuse. Furthermore, 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 has also been made, to be honest with our advisor about the current state of our project. Cooperativeness is 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 as a way to make sure 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 quality 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 biggest 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 important facts in creating software that helps extend longevity. By providing documentation, others can continue work and iterate on versions after initial teams move on from the project. Additionally, documentation is incredibly important in industry to allow for 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 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 are 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**

This semester I have failed to accomplish as much as I would like. 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. Over winter break and next semester I hope to demonstrate this virtue by going above and beyond my responsibilities.

Lincoln Kness

Virtue Demonstrated: **Willingness to Take Initiative**

A virtue that I feel that I have demonstrated is willingness to take ideas/prototypes to consolidate 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 individually each of us would 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 struggle with this semester is documentation throughout the work I am doing. 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 more broad sense. Over the course of the next semester I hope to make strides in showing these virtues more.

Ben Johnson

Virtue Demonstrated: **Problem Solving**

Throughout this semester 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 setup until halfway through the semester, and even now it isn't used as effectively as it should.

Zach Rapoza

Virtue Demonstrated: **Keanu**

One virtue that I have modeled this semester is 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 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 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 this semester is 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, but multiple times I failed to deliver a finished prototype on time simply due to a lack of invested hours. This could prevent us from reaching our deliverables next semester, so I be sure to allocate more time to development for this class.

Cory Roth

Virtue Demonstrated: **Thoroughness**

One virtue that I feel I have demonstrated well in this semester is my ability to be thorough and attentive to this project. I have been tasked with taking the notes during meeting and making sure that this group has all the information it needs in order to be successful. I have attended all the meeting 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 that I have not demonstrated well is my time management. I have had a busy semester with other classes, so I have not been able to commit as much time into this project as I would have liked to. This is important to me because in order to create a good project, time is needed to be put into it. You need to but time into the work in order to be proud of it. I will be working on this virtue in the next semester by allocating more time weekly to this project and helping out more.

8 Closing Material

8.1 Conclusion

As of now, the work that this team has done has included but is not limited to the following:

- Created a game idea
- Developed a fully fleshed-out game design
    - World Environment
    - Biomes
    - NPCs
    - Mechanics
    - Story
- Prototyped main mechanics of the game
- Created a simplified scene of the game
- Researched and understood Non-Euclidean math
- Implemented basic rendering
- Created a framework for rendering sprites and assets

- Initial Library for math rendering
- Input Handlers

The goals of this project are to:

- Create a Non-Euclidean 2-dimensional game/rendering engine
- Design and develop a game to showcase the engine

The best plan of action to achieve those goals is:

- Render Non-Euclidean math
- Finish prototyping game mechanics
- Working game demo in Unity
- Implement base game in render engine
- Continue to implement auxiliary features to polish the game

The project is currently in the middle stages of development for the created design. The proposed solution has good design concepts for the game and a solid foundation for the rendering engine. There are several future plans for this project that deserve mention. First is starting to implement features in the render engine that the game design needs. Another task will be completing the library that will be used to render the Non-Euclidean math. In terms of game design, the future plans include combining prototypes to build a working demo in unity. This way, the design and enjoyability can be tested from a user needs and constraints perspective. Once these are both complete, the main task will be integration. Following these steps, the goals of our project can be achieved.

There is one primary constraint that limits the progress of this project: the technical complexity of rendering Non-Euclidean math. Transforming data between different geometry spaces is no trivial task and takes time to implement correctly. Given the requirements and knowledge of the team members, there is not much that could be done differently, given a different design. If team members had a stronger understanding of the math before starting this project, it could be done quicker, however, this was not required, nor was it expected.

8.2 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=1908.01742. [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].

8.3 Appendices

A: Personas and Empathy Maps

Personas

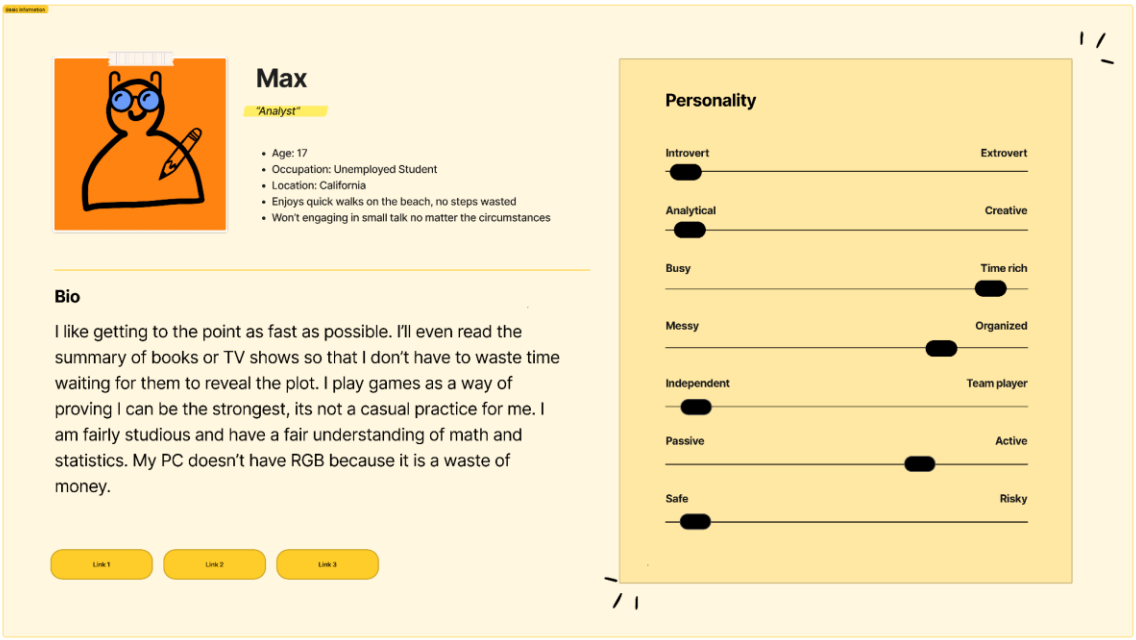

Figure 13 – Appendix A : Max User Persona

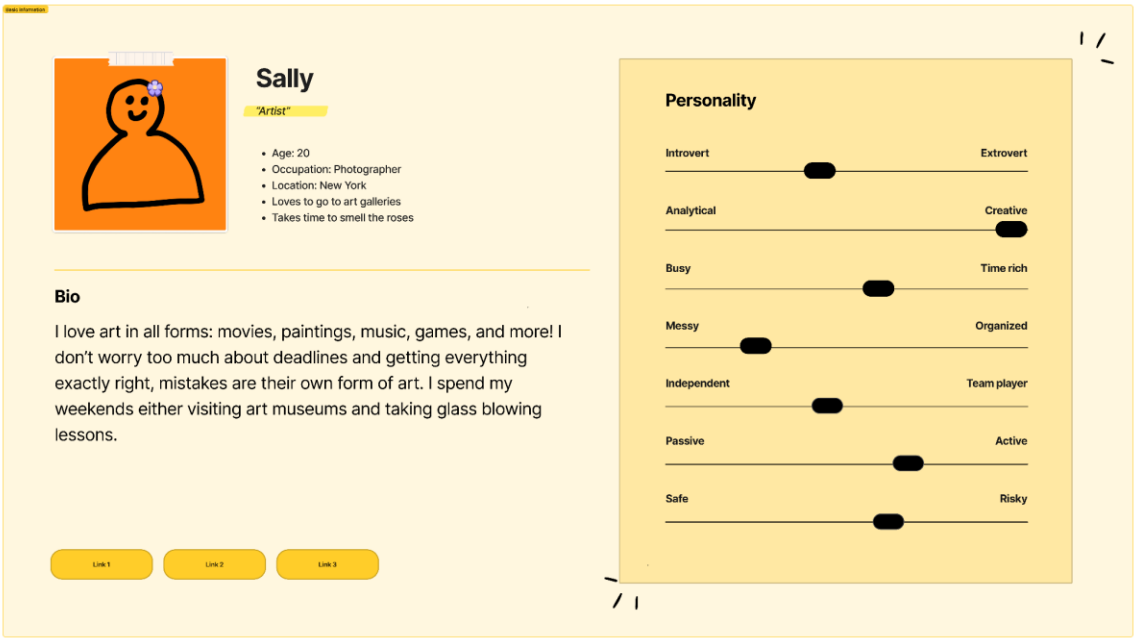

Figure 14 - Appendix A: Sally User Persona

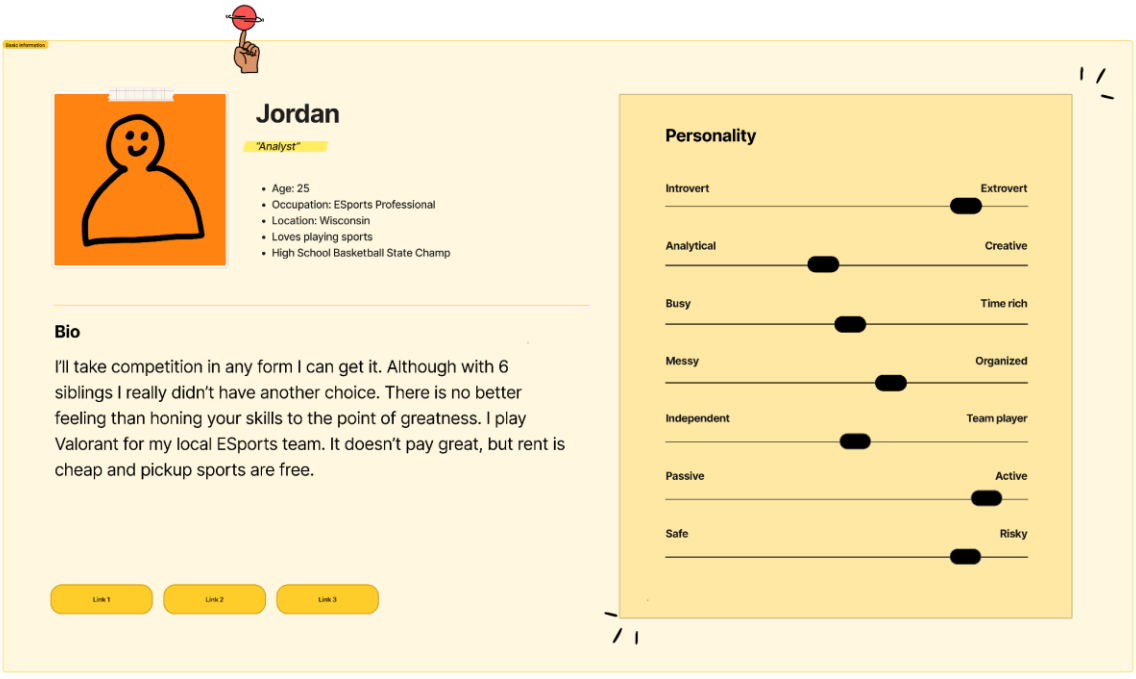

Figure 15 - Appendix A: Jordan User Persona

Empathy Map:

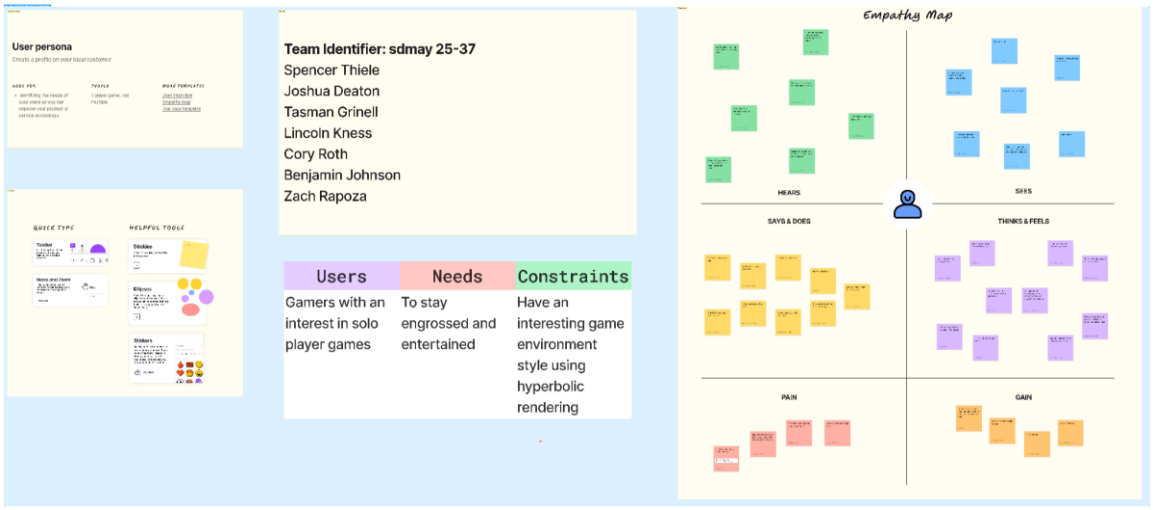

Figure 16 - Appendix A: Empathy Map

9 Team

9.1 Team Members

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

9.2 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

9.3 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

9.4 Project Management Style Adopted by the Team

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. The timeframe for this 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 that a shift can happen over the year from a heavy focus on waterfall to an increasing focus 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.

9.5 Initial 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 - Management Roles

9.6 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 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 of 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 Rendering 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	Rendering Engine 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 also is 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 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 even if you disagree with their idea, their point of view is still valid.

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___