4.2 Design Exploration

4.2.1 Design Decisions

The primary design decisions that were discussed and finalized were related to the software used to develop the project as well as the general high-level design of the game that will have to be implemented. In terms of tooling, software was determined based on ease of learning and standard tooling relative to common engines used in industry (in regards to prototyping for the game designs). Additionally, mathematical models were evaluated in a similar fashion, primarily with the accessibility and relevance to the high level game design. Finally, the high level game design was determined based on team interest as well as providing an entertaining idea for the game design.

Key Decisions

Software Usage(OpenGL and Unity)

- Chose to implement the render engine in OpenGL
  - Why
    - It is a cross-platform language that can be built on multiple devices.
    - It has good documentation, which makes it easy to learn and use libraries.
    - Regarding graphics languages, the learning curve is one of the lowest, So it will take less time to learn than others.
    - State-of-the-art software.
  - Why Important
    - This language will create the central system, the rendering engine.
    - 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.
    - The state-of-the-art system and other video games use it.
  - Why Important
    - The quicker features can be prototyped, the quicker it is to develop the game.
    - 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 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 Important
    - To implement non-euclidean math in our rendering engine.

        - Only selecting one field narrows the scope with a limited time frame.

Game Genre (Farming/Exploration)

- Choose to make a game in the farming/exploration genre
    - Why
        - Based on ideas generated for the main game, it was decided that this would be the most enjoyable to implement.
        - 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 good games.
        - Can implement features not seen in current games.
        - This genre typically has a good gameplay loop, a criterion of the project.
    - Why 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 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 to be of good quality, as one of the criteria needed was creating an enjoyable game. Focus was maintained on 2-D games, as a technical decision already made was that a 2-D 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 things 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

   - Shadows = dangerous
   - Switch between Euclidean and Non-Euclidean
   - Plants are light/shadow themed, shadow puts out nearby light, and lights give off a little light.
   - Night Pickups, you have to take your harvested plants and run them to a truck drop-off spot at night.

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 move on to the other one, 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 own 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, Roughlike, 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*. One of the main reasons it was chosen was the ability to utilize the non-euclidean space the best out 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 lacking in detail could be expanded. Also any additional clarification for 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 4.1 - 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 for an exciting feature to be added.

  This was a 5 of weight because it is the leading technical requirement, so it should be very important to the game.

  Euclid-Shot got the highest because a bullet hell where bullets travel in a non-euclidean manner makes the game unique and exciting to see. The light mechanic in *Lights Out* and seeing how lights would warp in a non-euclidean way was why it was given a four. 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 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

  This 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. The unique theme and gameplay loop ideas in *Lights Out* would make for a fun experience that would be 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 need to 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 shadow, 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 seem to 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 will need to 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 through managing memory, handling inputs, and drawing to the screen. The engine itself 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 4.2 - High-level System Diagram

The Rendering Engine

The rendering engine will process vectorized data from the game executable. It will compute the updated graphics that must be presented for a frame in a non-euclidean manner. It will send the frame to the laptop to be visualized. It will use resources that the ECS allocates. Its main features will be to draw the objects that the game engine wants to render and update the camera position. A top-down 2-D game view will be the game's main view, so a complete range of movement in the camera is not needed.

Entity Component System

The entity component system 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. It will also

be in charge of handling events that happen in the game. A significant feature that this system will have to manage is collision detection. Collision detection must be managed because most events in the game design are based on collision.

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 either send data to the Entity-Component system's 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.

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.

4.3.3 Functionality

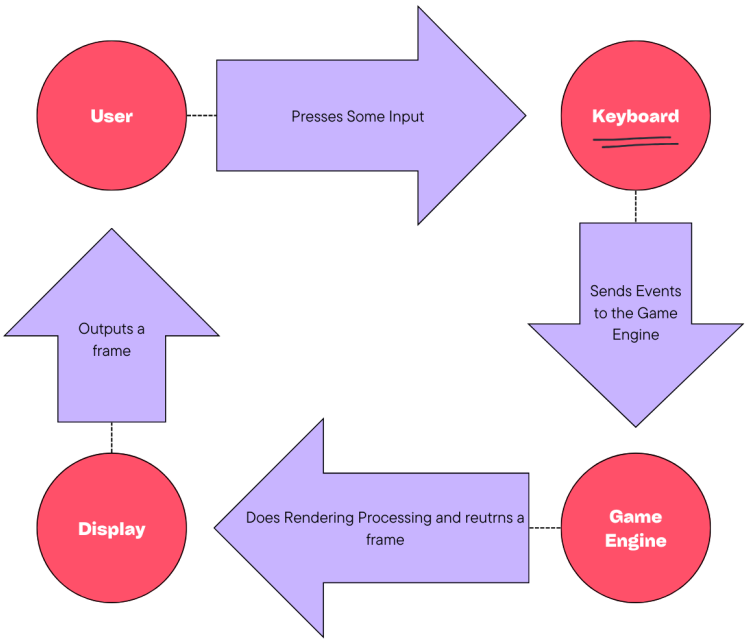


Figure 4.3 - User-System Feedback Loop

The user will interact with the proposed solution using a computer and some input device (keyboard or controller). When players launch the game, they interact with game elements such as moving the character, selecting inputs, and 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). It translates it into a series of visual frames.

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.

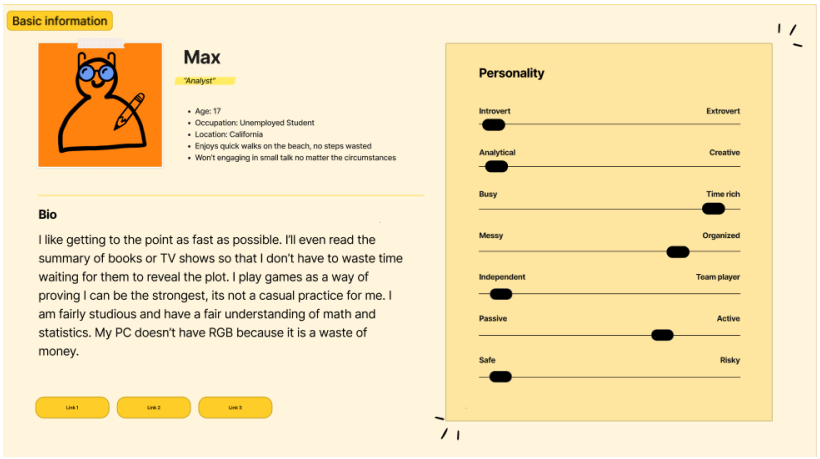
4.3.4 Areas of Concern and Development

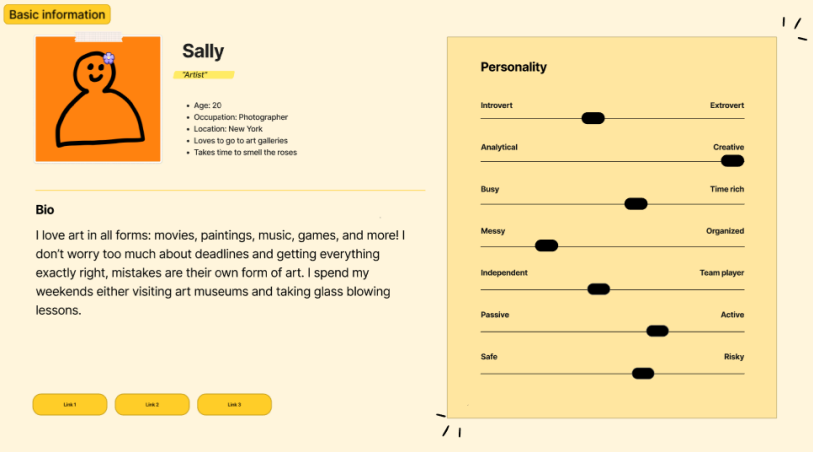
The current design will satisfy the requirements and meet the user's needs. There is not enough implemented to confirm this, but based on the progress made by each team, these requirements are believed to be feasible. Users' needs and technical requirements are being kept up based on what has been created.

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 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 asked to determine the feasibility of a project that seems completable within the time constraints. A meeting as a group will be set 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.





Included the personas to ensure user needs are considered when determining whether the design matches their 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 1 - 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. This is a positive considering this game might be on multiple different platforms. However, because of that, performance degradation might occur on specific platforms. An alternative choice could have been to develop in another language that only works on one platform with better performance. The choice was made that cross-platform ability 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 will good enough graphics cards / have a high-spec list like certain games do. The downside to this solution is that it may be harder to test those specs.

4.5 Design Analysis

Discuss what you have done so far, i.e., what have you built, implemented, or tested? Did your proposed design from 4.3 work? Why or why not? Based on what has worked or not worked (e.g., what you have or haven't been able to build, what functioned as expected or not), what plans do you have for future design and implementation work? For example, are there implications for the overall feasibility of your design or have you just experienced build issues?

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

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
- Have a framework for rendering engine
- 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 cannot be determined until the design is completed to give a valid answer to this question. What has worked is the proposed solution has good design concepts for the game, and there is 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 consist of combining prototypes to build a working demo in unity. This way, the design can be tested from a user needs and constraints perspective. No progress can be made to test the technical requirements until 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. It seems feasible that completing a rendering engine within the time frame is doable. It is also feasible to complete the game within the timeframe, However, integration may take longer than expected. That is because it is a complex issue and many game design features will have to be implemented in the engine that time may not be available.