Creating an Interactive Art Experience for Meow Wolf In Santa Fe, New Mexico

Nicolas Pingal  
*Worcester Polytechnic Institute*

Matthew Tucker Grim  
*Worcester Polytechnic Institute*

Griffen D. Spincken  
*Worcester Polytechnic Institute*

Follow this and additional works at: https://digitalcommons.wpi.edu/iqp-all

Repository Citation

This Unrestricted is brought to you for free and open access by the Interactive Qualifying Projects at Digital WPI. It has been accepted for inclusion in Interactive Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.
CREATING AN INTERACTIVE EXPERIENCE FOR MEOW WOLF IN SANTA FE, NM

An Interactive Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By

Nick Pingal
Matthew Grim
Griffen Spincken

October 12, 2018
Submitted to:
Professor Lauren Mathews, Worcester Polytechnic Institute

Daniel Bazo, Ph.D, Meow Wolf
Drew Trujillo, Meow Wolf

This report represents the work of [three] WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see: http://www.wpi.edu/Academics/Projects.
Authorship

Abstract

Executive Summary

1.0 Introduction and Background

1.1 The Rise of Creative Tourism

1.1.1 Creative Tourism in Santa Fe

1.2 Interactive Theory and Video Game Development

1.2.1 Different Types of Player Engagement

1.3 Interactive Experiences

1.3.1 Interactive Elements in Traditional Settings

1.3.2 Interactive Experiences in Nontraditional Settings

1.4 Meow Wolf

1.4.1 Origin of Meow Wolf

1.4.2 Meow Wolf’s Ethos

1.5 Objectives

2.0 Initial Game Concept

2.1 Determine the Initial Game Concept - Methods

2.2 Game Concepts - Results

2.2.1 Color Surfing

2.2.2 Starship Calamity

2.2.3 Hand Dancing

3.0 Design Document

3.1 Developing the Design Document - Methods

3.1.1 Design Document Creation

3.1.2 Digital Prototype Creation

3.1.3 Physical Prototype Creation

3.2 Design Document and Prototypes - Results

3.2.1 Design Document
3.2.1 Physical Prototype
3.2.3 Digital Prototype

4.0 Playtesting
4.1 Revise the Design Document Based on Playtest Feedback - Methods
  4.1.1. Design Playtest Structure
  4.1.2. Analyze Playtest Data
4.2 Prototype Feasibility Study - Results
  4.2.1 Demographics Overview
  4.2.2 Pilot Feedback
  4.2.3 Engineer Feedback

4.1.1. Design Playtest Structure
4.1.2. Analyze Playtest Data
4.2 Prototype Feasibility Study - Results
  4.2.1 Demographics Overview
  4.2.2 Pilot Feedback
  4.2.3 Engineer Feedback

5.0 Recommendations
5.1 Discussion
5.2 Physical Recommendations
  5.2.1 Pilot’s Physical Interface
  5.2.2 Engineer’s Physical Interface
  5.2.3 Other Physical Recommendations
5.3 Digital Recommendations

5.1 Discussion
5.2 Physical Recommendations

5.1 Discussion
5.2 Physical Recommendations

6.0 Conclusion
Abstract

The goal of this project was to design a video game in line with Meow Wolf’s unique aesthetic that broadens the range of experiences conveyed by Meow Wolf’s exhibits in Santa Fe, New Mexico. To do this, we worked to develop a design document and prototype video game for an exhibit that would potentially be implemented in Meow Wolf in the future. We developed a prototype of the fundamental game mechanics, and playtested the prototype with WPI students and Meow Wolf employees. Feedback from playtesting indicated that most playtesters found our game to be engaging, but one of the roles was significantly more engaging than the other. From our experiences playtesting the game and developing a prototype, we developed a set of recommendations for how to improve our game, and implement our game into one of Meow Wolf’s future exhibits, as a full room or a stand-alone arcade game.
Executive Summary

Meow Wolf is a Santa Fe company born out of the incredible success of their distinct and unusual artistic creations. Originally formed by a group of teenagers when their local community center shut down, much of Meow Wolf’s early work displays some of the maximalist aesthetic and overwhelming detail (Figure 1) which would come to define their most famous work to date, the House of Eternal Return (Aljinovic, 2016).

Figure 1: The Due Return

Despite the House of Eternal Return’s incredible success, Meow Wolf still maintains its independent, artist-focused ethos, and it is a for-profit company that focuses entirely on community benefit over individual profit (Meow Wolf, 2016).

In the design of new exhibits, the ideation process at Meow Wolf begins with the initial concept phase. In this phase each team member involved in the exhibit design proposes his or her ideas, and brainstorms new ones. These ideas are entirely unrestrained with no concerns about feasibility. After this, the team members discuss each idea and how it could be implemented, or whether or not it makes sense to use that idea for the current project. This process allows the team to narrow down the list of ideas to the most appropriate for the current project. Once they have narrowed down the list of ideas they begin to determine the best idea for the project from the list of remaining ideas. The final idea is usually decided upon by mutual agreement within the group. Once the initial idea has been decided upon, they then begin to develop the idea further, discussing various aspects of the idea such as art style, necessary components, materials, and controls. This process goes until the team agrees on the design. From here development starts and they create the exhibit. Our liaison at Meow Wolf, Dr. Danny Bazo, guided us through this process.

We were brought on to Meow Wolf to develop an experimental video game in line with their unusual aesthetic that pushes the boundaries of what has been done with control schemes
before. We began to develop an idea for our game by looking at an array of unusual sensors, and working through a number of preliminary ideas based on those sensors. During this process, we met with members of the Meow Wolf Technical Team, who advised us on how we could develop these ideas further. Over time, we settled on a design for a space game called Starship Calamity. In this game, two players have to navigate through hostile space in a broken-down spaceship. One player would play as the Pilot, directly navigating the ship itself, and one player would play as the Engineer, who would manage the ship’s remaining power by managing which features of the ship are enabled. We further developed this idea by creating a design document, which is a comprehensive document that details each planned aspect of the game.

After the creation of an initial design document, it was crucial to test our more experimental gameplay systems, to determine their viability. We chose to explore sections of our design such as player movement and communication. We initially began by developing a digital proof-of-concept, to test the feasibility of our movement systems.

This took the form of three separate programs created in Processing, a development environment used for simple prototyping of visual programs. Early internal testing proved that keyboard controls were inadequate for fully testing the viability of our control scheme, so we developed a much more elaborate physical prototype (Figure 2), with two distinct interfaces for the Pilot (left) and Engineer (right), controlled by Arduino microcontrollers.

Figure 2: Physical Prototype of Player Controls

Because we intended our game to be implemented in an arcade setting, we playtested our game with an audience that had never played our game before. We began by creating two extra levels for our game, designed to teach new players how the controls work. Due to the nature of our cooperative mechanics, each playtest involved a pair of players. Figure 3 shows two team members playtesting our game.
Each pair played through all three levels of our game, after which we interviewed them about their experience with the game, focusing on questions about the controls and the level of engagement of each player. After playtesting with WPI students, we playtested with Meow Wolf employees, set up at their offices for three days.

Playtesting confirmed to us the viability of our unusual control scheme and navigational methods. We found that 81% of players were able to successfully determine the direction that the objective was in using their distance from the objective alone. We also learned that our movement scheme was challenging but manageable, even for first-time players. Even with the Pilot relying on the Engineer for controls, 56% of pairs were able to successfully navigate to the objective. We believe this to be an acceptable percentage for a player’s first time attempting the game.

Playtesting also gave us some important insights about the physical interface of the game. We ran into some issues with the level of dependence our prototype had upon network communication. Interruptions in the wireless network would cause the game to freeze or the controls to fail. Therefore, we would recommend that the final game have one game system with
one embedded system that handles both interfaces. This dramatically reduces complexity and overhead while increasing the system’s reliability.
Acknowledgements

We would like to thank the following for making our project possible:

**Meow Wolf:**
Danny Bazo and Drew Trujillo, our project liaisons
Zevin Polzin and Wolves, for helping during ideation
The entirety of the Tech Team, for allowing us to use their equipment and playtesting our game
Katie Hannon, for helping us in the wood shop
Brandy Oleson, for allowing us to use the laser cutter

**WPI:**
Professor Mathews, our Project Advisor from WPI
Professor Fabio Carrera, the director of the Santa Fe Project Center
Professor Melissa Butler, our ID 2050 Professor
The 12 other WPI students participating in the Santa Fe Project Center, who playtested our game.
Authorship ii
Abstract iv
Executive Summary v
Acknowledgements ix

1.0 Introduction and Background 11
   1.1. The Rise of Creative Tourism 1
      1.1.1. Creative Tourism in Santa Fe 1
   1.2. Interactive Theory and Video Game Development 2
      Table 1: Important Game Development Terminology 3
      1.2.1. Different Types of Player Engagement 4
      Table 2: The Eight Aesthetics of Play 4
   1.3. Interactive Experiences 5
      1.3.1 Interactive Elements in Traditional Settings 5
      1.3.2 Interactive Experiences in Nontraditional Settings 6
   1.4. Meow Wolf 6
      1.4.1 Origin of Meow Wolf 7
      1.4.2 Meow Wolf’s Ethos 7
   1.5 Project Objectives 8

2.0 Initial Game Concept 8
   2.1 Determine the Initial Game Concept- Methods 9
   2.2 Game Concepts- Results 10
      2.2.1 Color Surfing 10
      2.2.2 Hand Dancing 11
      2.2.3 Starship Calamity 12

3.0 Design Document 13
   3.1 Developing the Design Document- Methods 13
      3.1.1 Design Document Creation 13
      3.1.2 Digital Prototype Creation 14
      3.1.3. Physical Prototype Creation 18
   3.2 Design Document and Prototypes- Results 20
      3.2.1. Design Document 20
      3.2.2 Physical Prototype 21
      3.2.3 Digital Prototype 22

4.0 Playtesting 25
   4.1 Revise the Design Document Based on Playtest Feedback - Methods 25
1.0 Introduction and Background

Creative tourism, as defined by the United Nations Education, Science, and Cultural Organization, or UNESCO, is the use of tourism to support cultural and artistic endeavor within an area (UNESCO 2018). The rise of interactive experiences is part of the Creative Tourism movement. The demand is rooted in the fact that people wish to immerse themselves in the art and culture of an area (Richards, 2011). A report by the University of New Mexico Bureau of Business and Economic Research found that tourists desire more interactive experiences (Mitchell, 2014), as seen by how the number of interactive opportunities in Santa Fe alone has increased by at least 600% since 2013 (OECD, 2014). This phenomenon has been growing steadily for the last twenty years and has caused the rise of different kinds of interactive experiences around the world such as Boda Borg in Sweden and other “escape room” activities in the United States (Bilis, 2015).

1.1. The Rise of Creative Tourism

Creative tourism is a form of tourism centered in creative and cultural involvement. Creative tourism is travel with an emphasis on engaging and authentic experiences, through active participation in the culture of the area (Bonetti, 2014). This wide definition contains a spectrum of activities that involve visitors in the culture of an area such as taking classes or actively participating in artistic experiences (Richards, 2011). This is in contrast to more traditional, static forms of cultural experience that are much more passive, such as viewing and buying works of art. Viewing of art and culture is better defined as “creativity as background” while creative tourism would better fall under the definition of “creativity as an activity” (Richards, 2011). Creative tourism allows visitors to experience culture by participating as they visit.

Creative tourism is an effective tool in the preservation and development of culture and art within an area (UNESCO, 2018). Santa Fe is internationally recognized as a “Creative City” by UNESCO. With this title comes the understanding that creative practices and sharing of art and culture will thrive within the city.

1.1.1. Creative Tourism in Santa Fe

Santa Fe joined the list of globally recognized creative cities in 2005 when it was added to the UNESCO Creative Cities Network or UCCN for short. This program recognizes and promotes cooperation between major centers of art and culture. To be invited to join the network, a city must recognize creativity as a major factor in their development (UCCN, 2018). Santa Fe is a city that thrives on its rich culture, propagated by supporting its world-renowned artistic community (Lee, 2007). Interactive creative experiences are a large component of the city of Santa Fe and this is evident by the many possible ways one can become involved in the culture of the city. An example of a major creative tourist event in Santa Fe would be Zozobra, which invites tourists to become involved in Santa Fe’s culture and history. This event involves burning
a 50-foot-tall effigy to expel all of the misfortune and woeful deeds accumulated over the past year (Edge, 2017). The festival demonstrates how distinctive Santa Fe’s own unique culture and traditions are, even among the other cities in the UCCN.

1.2. Interactive Theory and Video Game Development

Video games and interactivity go hand in hand as video games represent a merging of technology and art in a way that facilitates interactive experience. As we are developing a video game, it is important to understand some elements of game design, specifically how players will engage differently with different games. Listed in Table 1 are terms that will help to understand the rest of the section. To help to clarify the meanings of these terminology, examples will be given from the classic arcade game Pac-Man.

![Figure 1: Pac-Man](Pittney, 2009)

Pac-Man, originally released in 1980, has gone on to be the best-selling coin-operated arcade game of all time. The widespread popularity of Pac-Man makes it an excellent candidate for understanding video game terminology. In Pac-Man, the player controls the titular character (shown in yellow in Figure 1), moving through a maze to “eat” dots and four uniquely-colored enemy ghosts that would cause Pac-Man to lose a life (Pittney, 2009).
Table 1: Important Game Development Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Game Mechanics</td>
<td>Game mechanics are methods invoked by players characters, non-player characters, and objects in the game for interacting with the game world.</td>
<td>Pac-Man’s speed, the movement patterns of each ghost, the point values assigned to each action.</td>
</tr>
<tr>
<td>2. Core Mechanic</td>
<td>A core mechanic is the essential play activity players perform again and again in a game. It is still a mechanic.</td>
<td>Pacman, a game in which the main object of the game is to eat dots.</td>
</tr>
<tr>
<td>3. Dynamics</td>
<td>Larger systems composed of multiple mechanics.</td>
<td>The way Pac-Man must dodge in between each ghost’s movement pattern, the way the game becomes more difficult over time.</td>
</tr>
<tr>
<td>4. Aesthetics</td>
<td>The ways players engage with games. This does not refer to the game’s visuals in this context.</td>
<td>Challenge and Fellowship are the main aesthetics of Pac-Man.</td>
</tr>
<tr>
<td>5. Engagement</td>
<td>The player’s commitment to the interactive activities. Engaged players are focused on the gaming activities.</td>
<td>See Table 2. Section 1.2.1</td>
</tr>
<tr>
<td>6. Game engine</td>
<td>A game engine is a framework for game development that supports and brings together several core areas.</td>
<td>Pac-Man was created before game engines were under wide use for game development. Game engines under current use include GameMaker, Unity, Unreal Engine, Monogame.</td>
</tr>
<tr>
<td>7. Player Avatar/Player Character</td>
<td>An in game character that represents and is controlled by the player.</td>
<td>Pac-Man himself.</td>
</tr>
</tbody>
</table>

(Sicart, 2008)\(^1\), (Salen, 2004)\(^2\), (Hunicke, LeBlanc, Zubek, 2004)\(^3,4\), (Oksanen, 2017)\(^5\), (Unity, 2018)\(^6\), (Adobe, 2018)\(^8\), (Christensson 2009)\(^7\), (Pittney, 2009)\(^EX:1, 2, 3, 7, 8\)
1.2.1. Different Types of Player Engagement

It is important to understand the different ways players engage with games and how different mechanics and methods convey different experiences to the player. The predominant framework for understanding the ways players engage with games is the MDA framework, short for mechanics, dynamics, and aesthetics. Each level of this framework is composed of the previous levels: mechanics are the very base-level game systems, dynamics are composed of multiple mechanics and form the overall gameplay systems, and aesthetics are composed of multiple dynamics and define the way players engage with the game. Table 2 explains the eight aesthetics of play (Hunicke, LeBlanc, Zubek, 2004), (Portnow, 2012).

Table 2: The Eight Aesthetics of Play

<table>
<thead>
<tr>
<th>Aesthetic</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensation</td>
<td>Impressive visuals or sound, game as sense-pleasure</td>
</tr>
<tr>
<td>Fantasy</td>
<td>Immersion in another role, game as make-believe</td>
</tr>
<tr>
<td>Narrative</td>
<td>Involved story, game as drama</td>
</tr>
<tr>
<td>Challenge</td>
<td>Overcoming challenges for the sake of the challenge itself, game as obstacle course</td>
</tr>
<tr>
<td>Fellowship</td>
<td>Participation in a community, games as a social framework (Fellowship is not necessarily cooperation, competitive community elements still fall under fellowship)</td>
</tr>
<tr>
<td>Discovery</td>
<td>Urge to explore and discover secrets, game as uncharted territory</td>
</tr>
<tr>
<td>Expression</td>
<td>Customization, ability to make unique choices, game as self-discovery</td>
</tr>
<tr>
<td>Abnegation</td>
<td>Repetitive elements, focused more on relaxation, game as pastime</td>
</tr>
</tbody>
</table>

(Hunicke, LeBlanc, Zubek, 2004), (Portnow, 2012)

Analyzing Pac-Man through this framework will help to clarify how each level of MDA fits together. The mechanics of Pac-Man include eating dots, gaining points, and other basic
elements like Pac-Man’s movement speed and the conditions for spawning a fruit target. The dynamics are how these low-level systems come together, such as the ways the player has to dodge around each ghost’s unique pursuit pattern, the way the game becomes much more difficult as time goes on, and the conditions for generating a fruit. Finally, the aesthetics are the primary ways players engage with the game: in Pac-Man’s case, like many arcade games, the main aesthetics are challenge and fellowship, created by dynamics such as the high score table and the increasing difficulty of the game.

Fundamentally, players and developers approach games from the opposite ends of the MDA spectrum: players mainly experience the game through its final aesthetics, while developers mainly focus on creating mechanics. As such, developers must be mindful of the end experience they wish to create, to ensure that no mechanics or dynamics contradict the game’s main aesthetics (Portnow, 2012).

1.3. Interactive Experiences

The prominence of creative tourism has led new and established organizations alike to develop interactive experiences to meet the existing demand. The myriad approaches to interactivity and varying levels of success from these organizations are key to understanding the unique space our sponsor occupies within this field.

1.3.1 Interactive Elements in Traditional Settings

Traditional museums in the United States have been on the forefront of creating real-life interactive experiences since at least 1969, when the San Francisco Exploratorium first opened (Fernandez, 2000). The first of this kind of interactive museum in the United States, the San Francisco Exploratorium was very influential, and spawned a wave of other similar “science centers”. Studies from these early science centers have shown that children are much more willing to participate in interactive experiences than adults, enhancing the retention of information from the museum through incorporating interactive elements. Though these early science centers predate the creative tourism movement by about 30 years, their studies of interactivity are major formative works in this field (Caulton, 1998, Fernandez, 2000). Other types of museums have tried to incorporate interactive elements into their exhibits, finding economic and educational benefits. A study by the Royal Ontario Museum found that their Asian galleries’ new visitor rate increased from 57% to 63%, after incorporating more interactive elements, such as a scavenger hunt. Additionally, a survey given to gallery visitors found that the number of visitors that said that their visit was “exceptional” increased from 63% to 77% after incorporating interactive elements (Carding, 2015).

One of the main challenges for traditional organizations has been implementing interactive elements in a way that does not conflict with the desired message or distract from the information, seeking to inform rather than entertain (Adams, 2004). To this end, a number of museums have created new exhibits to discover the best methods of conveyance. A prominent example of this is the New York Hall of Science’s Human Plus Exhibit. The exhibit, intended to
inspire an interest in engineering in children, was composed of thirteen different activities, all themed around improving the lives of the physically disabled through engineering. These included video games played with unconventional control mechanisms and design challenges involving simple parts. A study of this exhibit found that the different elements of story, interactivity, and message could potentially compete with each other when trying to educate individuals, finding that specific methods of interactivity would convey different elements of the exhibit’s core message to visitors (Wasserman, 2015). Another exhibit was designed to highlight the cultural richness of Funchal, Portugal, and used light interactive elements such as gesture-controlled presentations and digital “books”, and an interactive section of the floor that conveyed the importance of transportation in the city’s history. Visitors found these interactive elements to be more engaging, which led to more focused engagement and a much greater retention rate of information (Campos, Pestana, Jorge, 2011).

1.3.2 Interactive Experiences in Nontraditional Settings

New organizations have also emerged, seeking to provide more exciting and immersive experiences to visitors, for the purposes of entertainment instead of education. The most popular of these experiences are escape rooms, which have grown exponentially in popularity within the last four years: within the USA alone, the number of active escape room companies has increased from 22 in 2014 to over 1900 by 2017 (Spira, 2017). Escape rooms provide visitors with a challenge: escape the room they are “locked in” before time is up, which involves solving puzzles and uncovering a story. Of course, other, related types of interactive experience exist, which provide different kinds of experiences to visitors. Some experiences like 5 Wits are inspired by Indiana Jones (Nicholson, 2015), focusing on using elaborate scenic design and special effects to provide a linear, story-driven experience to visitors. Other groups like Boda Borg, based in Sweden, provide visitors with multiple independent challenges with a wider scope than just escaping a room. Boda Borg predates the other organizations significantly, originally founded in 1995, but its first American location opened in 2015, seeking to capitalize on the demand for creative tourism opportunities in the United States (Bilis, 2015).

These kinds of organizations have been exceptionally successful because they provide a new kind of experience, compared to more traditional museum settings: their immersive qualities are an incredible draw for audiences (Kolar, 2017). Experiences like those provided by Boda Borg engage players through the aesthetics of fellowship and challenge, replicating the dynamics used in video game in a real-life setting to provide visitors with an engaging, challenging experience (Nicholson, 2015).

1.4. Meow Wolf

Our sponsor, Meow Wolf, has made great strides in pushing the boundaries of real-life interactive experiences by using a variety of colorful and interactive elements to immerse their visitors in the feeling of entering another world (Peter, 2017). A massive success, Meow Wolf had almost 400,000 visitors from across the world in their first year alone (Rodgers, 2017). One
of the central appeals of Meow Wolf is the wide range of interactive elements in their creations, as visitors are allowed to touch and interact with whatever elements of these displays they like (Meow Wolf, 2018).

1.4.1 Origin of Meow Wolf

After Santa Fe’s local youth center, Warehouse 21, closed in 2008, a group of 20 teenagers created an art collective named Meow Wolf, formed from a desire to continue working together on creative projects (Aljinovic, 2016). Their first location was a closed-down barber shop that they remade into their first exhibit known as *Biome Neuro Norb*. Meow Wolf later put on a variety of different shows and exhibits such as *AutoWolf*, which incorporated taking apart a Volkswagen Bug and creating into something different out of the pieces of the car. Their first project sponsored by another organization, the Contemporary Arts Center of Santa Fe, was *The Due Return* (Figure 2). This exhibit was the very first of their works to be designed from the ground up with an elaborate narrative integrated into their distinct, maximalist visual style. This fusion of narrative and design would prove to be incredibly successful for the fledgling group, bringing in over 20,000 visitors to the exhibit over its four-month run (Irwin, 2011).

From this point forward, the art collective became a company, and acquired a defunct bowling alley with the financial support of Game of Thrones author George RR Martin (Meow Wolf, 2016). They created their first permanent installation known as the *House of Eternal Return*, which brought in 400,000 visitors in its first year, and continues to bring in even more visitors each year (Ortega, 2017).

1.4.2 Meow Wolf’s Ethos

Meow Wolf’s mission is key to understanding their unorthodox aesthetic. Their mission is to provide artists with proper compensation for their work, acting as a for-profit artist’s coalition, focused on enriching the entire art community instead of just their own company (Meow Wolf, 2016). This mission is reflected in their incredibly broad, eclectic visual style: when asked about Meow Wolf’s aesthetic, Drew Trujillo, Meow Wolf’s current technical
director, described it as maximalist and all-inclusive, designed to make the styles of hundreds of different artists all feel like part of the cohesive whole.

The culmination of all these different styles of art can be seen in their 2016 exhibit *The House of Eternal Return*, which depicts a two-story Victorian-style house that blurs the lines between fantasy and reality by transporting visitors through portals to different dimensions (Meow Wolf, 2016). The exhibit uses the unassuming nature of a regular household to hide portals that lead to a chimera-like world ranging from a magical forest to an alien cave. There is no set path to follow and no map; visitors are encouraged to be curious and explore the exhibit and discover the story behind it (Wilson, 2016). This new form of art merges the work of over 150 artists into one whole exhibit, embodying the Meow Wolf aesthetic. All the artists involved in the project are paid double their counterparts in major urban areas (Solman, 2018).

### 1.5 Project Objectives

The goal of this project was to design a video game in line with Meow Wolf’s unique aesthetic that broadens the range of experiences conveyed by Meow Wolf’s exhibits. To do this, we accomplished the following goals:

1. Determine the initial concept of the game.

2. Develop a design document for the game.

3. Revise the design document based on feedback.

Our project focused specifically on developing a video game for Meow Wolf, combining technology and art to create a unique visitor experience.
2.0 Initial Game Concept

2.1 Determine the Initial Game Concept - Methods

Upon our arrival at the Meow Wolf production facility we were introduced to our project supervisor Dr. Bazo, a designer/developer for the Meow Wolf technology team. After that, we watched the Meow Wolf: Origin Story documentary. Because this film had not actually been released it was impossible to actually view it until we had arrived at their facility. From this we were able to better understand Meow Wolf’s central driving ethos of unrestrained creativity. As a result of this ethos, Meow Wolf’s ideation process, used by all of their artists and developers alike, allows for complete creative freedom. We had two requirements placed on our project: we were to design a game to be potentially incorporated in the arcade in the House of Eternal Return, and our game must use unconventional input, beyond just using standard buttons or game controllers. Because the future location for our game was decided by our advisor, we focused more on creating stand-alone game concepts instead of incorporating.

We started the ideation process by looking at different hardware components that could potentially serve as our unusual form of input. We primarily looked on Adafruit.com, as recommended by Dr. Bazo, a website with many different and unique hardware pieces, such as flow rate sensors, magnetic field sensors, and distance sensors (Adafruit, 2018). Each team member then developed two to three very rough game ideas using these sensors, exploring the range of how it was possible to implement these sensors. After we had created early ideas for games revolving around these sensors, Dr. Bazo brought us on a tour around the technical department to show us what other employees were working on to help us think of unique ways to use the unique sensors. We were shown a piece of hardware that was the inspiration for what would eventually become the core component of our final concept. The piece of hardware was a board of buttons without a full exterior case, leaving the wires behind the buttons exposed. A discussion we had with an employee who was working on a piece of hardware led us to our central concept which was a game about plugging in the buttons that control the game.

We returned to our ideation process with this idea. We continued to develop these ideas, eventually creating five rough, unique ideas for a game. We then met with Dr. Bazo two to three times a day to discuss which ideas we had agreed upon would make the best games, this process lasted 3 weeks. He gave us feedback on each idea, including potential alternate directions to take each design, as well as potential issues that could arise during development. The group, with Dr. Bazo’s guidance, brought the total number of designs to focus on down to three. The three of us decided to each work on an idea on our own and further develop them.

We each spent several days developing these ideas until we reached a point where we had created the core concepts of each game. We then met with Dr. Bazo, who brought two other Meow Wolf designers to this meeting. The first was Zevin Polzin, a programmer, who brought up gameplay and technical concerns. The second team member was a programmer who goes by the name Wolves, who brought up concerns of accessibility and durability. Other topics brought up during this meeting were materials, narrative, visual design, and sound design. The three of us
then worked as a team on each idea individually, attempting to address the Meow Wolf team’s concerns. We spent seven days on the process to help ensure we addressed as many of these concerns as possible. Once we had reached a point where we all agreed the designs were developed enough we met with Dr. Bazo and decided on the final game design: Starship Calamity. This decision was based on several factors, such as overall uniqueness, input methods, original gameplay concepts, complexity, and which idea we thought was the most ambitious.

2.2 Game Concepts- Results

Initially we created five basic concepts for our game. The first was a two player space game, where one player controls the ship, and the other player chooses which controls are active by physically plugging them in, similar to a modular synthesizer. The second was a color matching game where the player would change red, green, or blue values of a color by increasing or decreasing the distance to a time-of-flight distance sensor with your hands. The third idea was a game where the player would control an alien spaceship abducting cows. The fourth idea was a racing game played on a strip of LEDs, similar to slot cars. The fifth and final idea was a game about gliding, controlled with just one of the distance sensing sensors from the original color matching idea. We trimmed these initial five designs down to three through a series of ideation meetings with Dr. Bazo, where we decided to focus on the three ideas that used more experimental hardware: the color matching game, the gliding game, and the space game.

2.2.1 Color Surfing

This design was inspired by our research into unconventional sensors that could be used as game inputs. The idea for this game was based on a time-of-flight sensor that would measure how far an object was from the front of the sensor. Three of these sensors would be mounted on a panel in front of the player. The original sketches for this game idea can be viewed in Appendix I, and a concept render of the Color Surfing arcade cabinet can be seen in Figure 3. By holding his or her hands over the sensors, the player would be able to manipulate the Red, Green, and Blue color components of his or her character. The character existed on a colored track and would progress only when its color matched that of the track. Player movement would be based on how well they match the color of the track they are touching and this would provide forward acceleration. At set intervals the player would be rewarded with additional time to continue progressing. As the game continued, changes in the color track would become more varied and
would change more rapidly and make progression more difficult. The player’s high score would be the total distance they had travelled before running out of time.

This track could also have been displayed on either a traditional monitor or a less conventional display such as an individually addressable LED strip. Basic mechanics were sketched out and a model of a potential cabinet was made using SketchUp.

We decided not to follow up on this design after pitch feedback suggesting that we take the game in far more complex directions that we had originally intended. Attempts to add additional complexity to the game such as putting it in two or three dimensions as opposed to one just made the game more difficult to understand instead of more engaging and entertaining.

2.2.2 Hand Dancing

The original concept for Hand Dancing was to develop something that uses as few inputs as possible. The designs that became Starship Calamity and Color Surfing were to use a variety of unusual inputs to control multiple systems simultaneously, so Hand Dancing was developed to use a single time-of-flight sensor. The player would raise and lower his or her hand above the sensor to control a glider, which would aim up and down, matching the movement of the player’s hand. The player would have to use this control scheme to navigate a glider through a series of abstract obstacles that would appear in time with the music, adding an element of rhythm matching to the gameplay. The player’s score would be exclusively determined by how long they could survive without hitting an obstacle. We decided not to continue with this concept because it was not as ambitious or exciting as the other games we pitched.
2.2.3 Starship Calamity

Our final design, Starship Calamity, is 2D game about piloting a damaged spaceship through asteroids filled hostile space. Due to how broken-down the ship is, only a certain number of the ship’s systems can be activated at once. Because of this, the game is played by two players split between two different roles: the Pilot, who controls the ship directly, and the Engineer, who manages which systems of the ship are active at what time. This game’s unconventional input are the “power cables”, which the Engineer must plug into various patch bays to “engage” these systems for the Pilot’s use. Only with careful communication between the Pilot and the Engineer can the players find the wormhole and escape hostile space. Figure 4 shows an early design concept for the Starship Calamity arcade cabinet. We decided to pursue this idea because it was our most ambitious idea, and our liaisons liked the concept of an overwhelming control scheme that cannot be managed by one player, and our thought that it fit in well with Meow Wolf’s maximalist aesthetic.
3.0 Design Document

3.1 Developing the Design Document- Methods

Once we decided on using the Starship Calamity concept for our final design, we began to work on a comprehensive design document. To further expand on these ideas, we also developed a digital prototype to test the feasibility of the cooperative movement systems.

3.1.1 Design Document Creation

A game design document is a highly descriptive, living document that records all aspects of a game. The documents are divided into sections such as story, characters, level information, gameplay, art, sound and music, user interface and game controls, and accessibility. Not all game design documents are written in this order or have the same aspects. The document is kept up to date as changes to the design are made.

We began by outlining the design document, deciding the order of each particular topic, choosing to focus more on mechanics than visuals and music or other sound effects. We included an additional section detailing a range of possibilities for hardware and physical components. As we agreed upon the various aspects of the game, we recorded them in the design document, and implemented them into our digital and physical prototypes. Our design document was focused on the most important mechanics of our game; we also created a recommendations section at the end for our more elaborate ideas and ideas that we could not implement within our time in Santa Fe.

We spent three weeks focusing on the initial draft design document, meeting with Dr. Bazo every day during this period. During these meetings, instead of pitching multiple game concepts and getting feedback on each of them, we presented the current state of just the Starship Calamity, and all of its aspects in detail. These meetings helped us to refine the mechanics of the game. An early concern brought up by Dr. Bazo was that the Engineer would not have enough to do, only really being able to play reactively and only change things when the Pilot asks. To make the Engineer more proactive, we developed extra abilities that the Engineer could use, as well as a radar system that would allow the Engineer to communicate more meaningful information with the Pilot.

During these meetings, a member of the Meow Wolf technical team who goes by the name Wolves brought up several concerns about materiality. Wolves’s first concern was about accessibility, given how unusual our Engineer’s interface was. To this end, we measured existing arcade cabinets and worked with Wolves to find appropriate dimensions for wheelchair accessibility for our game. Wolves’s other concern was durability, given how roughly Meow Wolf exhibits are handled. For this concern, we designed very durable power cables with Dr. Bazo.
During these meetings, the overall direction of the game changed significantly, and we decided it would be best for us to create a prototype of our core game systems to properly evaluate their feasibility, and to remove any ambiguity about how the game will function.

3.1.2 Digital Prototype Creation

To assist with the creation of our design document, and to evaluate the feasibility of our more experimental mechanics, we developed multiple software and hardware prototypes.

Our first digital prototype was made in GameMaker Studio 1.4, a simple game engine that the team already had experience using. This version was only controlled by one player on a single computer, intended to just test the feasibility of our movement and collision systems, and to remove any ambiguity about how we wanted those systems to function. We recorded the movement system in the design document.

When creating the prototype in GameMaker Studio, first we created the sprites for each object, which included the ship and asteroids. Then we created the bounds for the game area, by placing impassable walls outside the area the player sees. The screen did not scroll as the ship moved. We then populated the play area with randomly positioned asteroids that had a random speed in a random direction every time the game started. After we implemented the player and asteroids into the game, we implemented controls. The controls were simplistic, with the W, A, S, and D keys imparting a speed to the player in the up, left, down, and right directions respectively. Next, we implemented the physics systems, to experiment with movement that had high amounts of momentum, utilizing GameMaker Studio’s built in physics systems. This system caused the ship and asteroids to bounce off one another, and for the ship to bounce off the arena’s boundaries. The movement system was also changed so it imparted momentum to the player and asteroids instead of speed, because using speed would not allow us to use physics systems on the player and asteroids, a quirk in GameMaker Studio’s physics system.

The next prototype involved 3 computers. We used a development environment called Processing, specifically recommended by Dr. Bazo, because it allowed us to develop the prototype much faster with its built-in graphics libraries. Two computers had Engineer and Pilot interfaces, and the third had the game screen. Figure 5 shows the information sent between each program running in the final prototype we used for playtesting.
This diagram shows the communication between each of the three game programs, represented by the boxes. These three programs, the Engineer’s control screen, the Pilot’s control screen, and the Game screen, were linked together with a communication protocol called Open Sound Control (OSC). Dr. Bazo recommended we use this network protocol because Processing had it already built in, so it was very straightforward to integrate it into our prototype. Other game engines would have had to use more complex communication protocols to properly network the three programs together.

The first step in creating this prototype of the game was finding a way to have a background scroll in two dimensions. This scrolling effect allows for the illusion of movement even though the player is always located in the center of the screen. This was originally attempted using images of a star background that would generate in a tile-like pattern and replace each other as they moved behind the player. After discussion with Dr. Bazo, we decided that it would be best to draw a black background with dots for stars rather than use static images for the background. Rather than moving the whole background image we can simply move the stars to create the illusion of movement. This system uses a large array of star positions and as they get far enough away they are moved to the opposite edge of the screen to make sure that processing power was not wasted.

After finishing our work on the scrolling background, we added asteroids, which scrolled across the screen using the same scrolling code as the background. These asteroids were drawn in a very similar way to stars, except they were much larger and would always appear in a range of browns, as opposed to the stars’ more varied colors. Asteroids were set to disappear when the player moved sufficiently far away from them, and to reappear in directions the player was moving, with a percentage to cause more asteroids to appear as the game went on, conveying the feeling of moving deeper into an asteroid field. Figure 6 shows how the game screen looks with...
asteroids, stars, and the player’s ship implemented. This gave the appearance of asteroids coming into view as the players move through space, rather than just appearing unexpectedly.

**Figure 6: Starship Calamity Normal Game Screen**

After we implemented asteroids, we developed the game’s physics system. Essentially, each in-game object that the player could interact with has a mass and velocity, which determines the amount of momentum the object has. Both the player’s ship and the asteroids were given these properties, which determined the speed and direction of each object after collisions. The only collisions we enabled would be between asteroids and the ship, to help optimize the program and simplify the code. Originally, asteroids were set to fly onto the screen at varying speeds, but this was changed to only allow for still asteroids to appear after early playtesting by the team. This change was made after some early internal playtesting. We found the game to be much less frustratingly random with stationary asteroids. This change also put more of an emphasis on strategy and planning.

After that, the game’s damage system was implemented. Using the code from the physics system, we set the program so that the ship would take much more damage when colliding with asteroids that were heavier. Additionally, if the ship was moving faster, it would take much more damage from collisions. If the ship took too much damage, the game would be lost and the player would have to restart.

The final system we implemented was the end goal, which was to reach a wormhole. The wormhole appeared as a large purple object that would appear far away from the player’s initial
position in a random direction. If the player touched the wormhole, they would win the game. The Pilot could not tell where the wormhole was, but the Engineer could see how far the ship was from the wormhole and therefore could detect when the ship was moving towards or away from it. The only way the Pilot could find the wormhole was by carefully communicating with the Engineer to locate it.

After implementing all of the basic functions of the game, Dr. Bazo recommended that we design three levels to use during playtesting to ease new players into the controls and test the feasibility of the controls. The first level just had the players navigating around a small line of asteroids to reach the wormhole, and was designed to teach the players the basics of navigation with our unusual control scheme. Figure 7 shows tutorial level one, along with the wormhole, the level objective. The second level was like the first, but the line of asteroids had small gaps in between each one, forcing the players to slow down and move carefully. This level was designed to teach players about the importance of moving slowly and deliberately with the ship. Figure 8 shows tutorial level two. The third and final level was what we had created already, with randomly generated asteroids and the Wormhole generated very far away from the player, intended to test the fairness of our random generation systems and the improvisational abilities of the players with our control methods. Figure 6 shows level three.
3.1.3. Physical Prototype Creation

Our first control interface for the game controllers consisted of the standard keyboards on our laptops. We used these because Processing had built in functionality to communicate with our keyboards. These were primarily used as placeholders to test the digital prototype while we built the other physical prototypes. We used the arrow keys for the Pilot’s movement and to turn systems on and off on the Engineer’s side. Because these were unsatisfactory after initial internal playtesting, we decided to create a more elaborate physical interface prototype. Figure 9 shows an early concept render of this interface.

It is important to understand what an Arduino is because all of the prototypes used them to interface with the computer, with the exception of the keyboard. An Arduino is an example of an embedded system where the processor runs one usually very specialized function (Professor Jarvis, Personal Communication, Dec 2017). It allowed us to use several different types of control methods for our prototypes. It also allowed us to build them without the need to permanently connect wires or other hardware pieces. Meow Wolf had many of these available for project use, and because of their functionality and availability, Arduinos were convenient for use in our physical prototypes.

The next piece of hardware we implemented was the joystick, a CH Products series III joystick. The joystick is capable of eight-directional movement. We used the joystick because it fit our control scheme and aesthetic of piloting a spaceship. The eight-directional movement allowed the player to fire one thruster for movement or turning, or two at the same time, allowing for movement while turning. Initially our joystick was mounted in an acrylic panel Meow Wolf already had a different joystick mounted in, used for testing. We removed that one and mounted our own. We could not use the joystick they had already mounted as it plugged directly into the computer and required specialized programs to run that we could not find. Our joystick would eventually be mounted in its own box with LEDs around it to indicate which functions were engaged at any given time, depending on the Engineer’s controls.

We also created the button interface for the Engineer’s controls while we installed the joystick. The interface was made out of 4 small buttons, shown in Figure 10. We used these buttons as a way to test if the Arduino was properly communicating with the computer, and to allow for further playtesting without the Engineer looking at the screen. We also installed the LEDs to indicate which directions were enabled on the joystick.
After the buttons, we decided to emulate the controls in a way more accurate to the final design. We removed the buttons and replaced them with wires that the Engineer would physically plug into the circuit board. This gave us a more accurate representation of the Engineer’s plug-based interface. Like the final design, we gave the Engineer fewer wires than he or she needed to power all the systems.

Our final prototype design consisted of two boxes, one with a joystick, and the other one with the plugs, and a computer for each box. Both boxes had an Arduino inside of them that sent signals from the controls to the computers. The box with the joystick was the Pilot’s box. In addition to the eight-directional joystick, shown on the left in Figure 11, the box has four LEDs,
two along each axis on either side of the joystick. These LEDs were connected to the box with the plugs.

The box shown on the right with the plugs is the Engineer’s box. It consisted of three one-quarter inch headphone jacks, and four one-quarter inch plugs. The plugs were located in the same layout as the LEDs on the Pilot’s box. Both boxes were the same dimension, and made with an acrylic top and wooden sides. The only difference between the two boxes is that the Pilot’s box has a wooden bottom with a four-inch lip in the front while the Engineer’s box had an acrylic bottom, with a two inch lip in the front. These lips were added to the boxes in case they needed to be clamped to a table. The lip on the Pilot’s box is bigger so we could use more clamps as we predicted it would need more stability, due to how much the joystick needed to be moved, compared to the Engineer’s plugs. We made the boxes out of wood and acrylic because Meow Wolf had these materials available on site, and we could quickly cut and assemble the parts for the boxes. We used the quarter-inch jacks and plugs because they were commonplace, gave tactile feedback, and were durable, easy to install, and easy to obtain. The electronic components, such as the wire and LEDs, were used because Meow Wolf already had all of the pieces on site, and readily available.

3.2 Design Document and Prototypes- Results

After we decided to focus on Starship Calamity, we fleshed out its design document, and built two prototypes. These prototypes included physical design and builds for the controllers, and iterations of the game on the software side. As we built them, the game was adjusted based on what we learned from that prototype.

3.2.1. Design Document

Our design document contains sections on all of Starship Calamity’s mechanics and systems, with a particular focus on the two roles of Pilot and Engineer and the division of information and controls between them. This document is intended as a set of guidelines for a team of developers to use to implement our game in the future. As such, we also included a set of recommendations in the design document, focused more on mechanics that would be interesting from a gameplay perspective, but are not essential to the core of our game. The design document can be read in Appendix D.
3.2.2 Physical Prototype

The physical controls for the two roles both used Arduino microprocessors to take input from electrical circuits such as the joystick and plugs. Communications between these microcontrollers and the main computer are handled by a USB connection.

Communication across USB is handled in a bitstream style where ones and zeros are sent to the computer in a line where they are interpreted (Arduino, 2018). Because the amount of information we send is relatively small, we send the information in a package that is the same size as a single keyboard character. A character or char is the smallest piece of data that can be sent across a USB connection using the Arduino interface (Arduino, 2018). This stream of information is made up of eight ones and zeros and is called a byte. Each of the eight positions within this byte is known as a bit and can either hold that value, one or zero. For our purposes because we only have four systems we only require four of the available eight bits and four will be unused by our program. Once the char has been transmitted to the computer over USB, Processing is able to look at each bit within the char to determine which systems are to be enabled. The order of bits are: up, down, left, right, with a 1 indicating an active thruster Thus, the code 0110 would tell the processing code that the down and left thrusters are to be enabled.

The physical interface for the Engineer is composed of four quarter-inch audio jacks in place of the plugs that would be used in the final game. These audio jacks are used to get a better test of the movement system than could be accomplished with a button interface. The diagram for this circuit can be found in Appendix F. Each port is connected to a digital pin of the microcontroller and the ground pin across a “pull down resistor” as shown in Figure 13. The purpose of this resistor is to make sure that the pin remains at zero volts while no voltage is applied to it. The jack is connected to the five-volt pin so that when the jack is inserted into the plug it raises the voltage of the digital pin to five volts. This tells the Arduino that the plug has been activated. Each input pin for plugs has an associated output pin that is connected to the LEDs in the Pilot’s interface. When a plug is recognized as activated it both sends a signal across USB to the Engineer’s digital interface as well as lights the corresponding LED to allow the Pilot to see what directions are enabled.

![Figure 12: Byte Structure](image)
The joystick that we were supplied was a spare that was available and was more than adequate for our purposes. The datasheet for this joystick can be found in Appendix E. This joystick does not actually have a way to interface with a computer out of the box. Internally it is composed of two sensors that each output a voltage from zero to five volts and represent the X or Y axis of the joystick's position. As seen in the code in Appendix A, the Arduino takes the voltage from these sensors in pins numbered A0 and A5 shown in Figure 14 and compares this to the assigned sensitivity variable. If the value exceeds the sensitivity threshold, the Arduino reports that that direction has been selected.

3.2.3 Digital Prototype

The digital prototype of Starship Calamity consisted of three sketches running across three different computers: the Pilot’s interface, the Engineer’s interface, and the game screen. Figure 15 shows the Pilot's digital interface.
The red indicators above each thruster display which ones can be activated. In this case, the Pilot’s joystick is positioned diagonally forward and to the right, and the Engineer has only engaged the rear thruster. Because the rear thrusters are engaged and the Pilot is attempting to use them, the display is green, showing that those thrusters are active. Because the right thrusters are not engaged, but the Pilot is trying to use them, the display is yellow, which should signal to the Pilot that he/she needs to request that the Engineer engage those thrusters.

If Pilot moves the joystick in a direction and the thrusters in that direction are engaged, the corresponding indicators turn green. Because the joystick can move diagonally, up to two thrusters can be engaged at once. There is a light above each indicator that shows whether or not the Engineer has engaged that system. If the system is engaged, the light is red, otherwise the light is dark. The top and bottom indicators show forward and reverse thrust activation, and the left and right indicators indicate torque is being applied to turn the ship.

Figure 15: Digital Pilot Interface

Figure 16: Digital Engineer’s Interface

Figure 16 shows the Engineer’s digital interface. The Engineer has 4 indicators. These indicators turn green when the corresponding system is engaged with the power cables. When the indicators are active, they turn on the red active light above the corresponding thruster display on the Pilot’s side, and power the joystick’s directions. The Engineer also has the power meter. If the power meter hits 0, then no systems can be activated at all, and the ship starts to lose health,
quickly ending the game. The Engineer has the health meter on his or her screen. The health meter is displayed as a percentage, and the game ends if it reaches zero. The Engineer can also see distance to the objective, but no direction is given, so the Engineer and the Pilot must communicate to locate the objective.

The game screen shows a starfield, and asteroids appear from all sides, with more appearing as the game goes on. The goal of the game in our prototype is to reach the wormhole that appears far from the ship’s starting position in a random direction, to escape the asteroid field and win.
4.0 Playtesting

4.1 Revise the Design Document Based on Playtest Feedback - Methods

The digital and physical prototypes were specifically created to allow us to test the feasibility of our mechanics, so we could produce recommendations for how to implement them in the future. To this end, we had WPI students and Meow Wolf employees playtest our prototype and give feedback on their experience.

4.1.1. Design Playtest Structure

The goal of our playtest was to determine how easily our complex control systems could be learned by players who had never experienced them before. Based on our timeframe, the amount of data we needed from each playtester, and the number of playtesters we could schedule, we determined that one-on-one interviews would be the best for obtaining the most valuable feedback from each player (Patton, 2017). We developed the interview script, deciding on seven specific questions that would give us the most valuable feedback about our game, based on a common set of generally useful playtest questions (Rockholz, 2014) and some questions we developed that address specific aspects of our game. One of our questions asks players which direction they felt the objective was in. This is because early internal playtesting found that it was easier to reach objectives that were towards the top of the screen compared to the ship’s starting position, compared to towards the bottom of the screen. We wanted to confirm this phenomenon with a larger sample. The final playtest script can be read in Appendix C.

Next, we began the signup process for our playtests. One week before the playtest sessions, we sent out a signup sheet to all of the WPI students working in Santa Fe at this time, allowing them to select their session time, and who they would pair up with. This signup sheet can be seen in Appendix N. After our playtesting with WPI Students, we playtested with Meow Wolf employees at their offices. We set up our game in the employee lounge at the Meow Wolf production facility, the most convenient place to find employees that were not otherwise occupied. We playtested at the Meow Wolf offices for three days.

Each playtest session was structured as follows: first, we gave each player a brief description of how the controls worked, without going into too much detail about them, to allow the players to discover the subtleties of our controls on their own. Then we had them play through all three levels of our game. While they played the game, we did not speak to them at all, so as to not influence their gameplay or alter their responses. Each pair played through each level once. During each step of this process, we took notes on any bugs that emerged that we had not seen before. Then, they were interviewed, using the interview script, and their responses were recorded in a spreadsheet.
4.1.2. Analyze Playtest Data

We focused our analysis of the data on issues that the players had with the controls. Other recurring issues we looked for included issues with visual and audio feedback, and the fairness of division of responsibilities between roles. From our question about player experience, we found that all of the answers to the question about prior player experience cleanly fell onto a scale of 1-5, so we coded those answers to that scale for analysis, where 1 corresponded to little to no experience with games, and 5 corresponded to extensive experience with games. We used this data to form a set of recommendations from the most common player feedback.

4.2 Prototype Feasibility Study

4.2.1 Demographic Overview

In total we had 32 people playtest our game for a total of 16 playtests. In total 22 were male, and 8 were female, and one was nonbinary. From our sample we were not able to observe a meaningful correlation and game experience. We had 9 Pilot playtesters from WPI, and 9 Engineer playtesters from WPI. Some students playtested both roles. From Meow Wolf we had 7 playtesters for both the Pilot and Engineer roles. The age distribution of all the playtesters can be seen in Figure 17. As more than half of our playtesters were students from WPI, the age distribution is heavily skewed in the 19-21 years old range.
We also recorded the Pilot’s and Engineer’s self-reported prior experience playing video games for each pair of players, on a scale from 1 to 5, where 1 was very little experience, and 5 was extensive experience. The Engineer’s prior experience did not make as much of a difference, as they had significantly less to do. We found that there was a very slight correlation between the Pilots’ prior experience and their win rate, but this was probably an effect of our small sample size. Figure 18 shows the distribution of Pilots’ prior experience with video games, and Figure 19 shows the win/loss ratio along these lines.
4.2.2 Pilot Feedback

A total of 16 people participated in our playtest in the Pilot role; 9 of these were WPI students and 7 were Meow Wolf employees. From our playtests we received valuable feedback on the controls and interface from people who had not played the game before. Of our 16 Pilots
33% of them referred to the controls as unintuitive or frustrating. The most common frustration from these Pilots was that the turning thrusters were too powerful and rotated the ship too much when activated for. Despite this, 53% of pilots found it satisfying when they finally figured out the controls and could fly the ship skillfully. Despite these concerns, 100% of Pilots felt like they had enough to do.

Early in internal playtesting we discovered that there was a possibility that it was more difficult to navigate the ship in directions other than up (towards the top of the display screen) because the ship would no longer be aligned with the joystick controls. Specifically, when the ship is oriented down, the Pilot’s “right” will correspond with the ship’s left, leading to player confusion. To explore this we separated the play results of those who were assigned upward objectives against those that were assigned southern objectives. Figures 20 and 21 show the difference in win/loss ratios between players who had to go up and players who had to go down. In total there were 6 playtests where the objective was to in the upward direction, and 9 when the objective was in the downward direction. By comparing the success of these two groups of players we were able to observe a difference in the win/loss ratios of the two groups, but we do not have a large enough sample to prove this with statistical certainty.

### 4.2.3 Engineer Feedback

A total of 16 people participated in our playtest in the Engineer role; 9 of these were WPI students and 7 were Meow Wolf employees. Our most concerning finding was that 50% of Engineers felt that they did not have enough to do compared to the Pilot. This was a known issue of our prototype going into playtesting, because we could not implement all systems we designed to make the Engineer’s gameplay more involved. See Appendix D for a full list of features in the design document. These Engineers wanted to have some form of additional control over what was going on in the game, instead of reacting to what the Pilot wanted. Furthermore, we observed that the Engineer’s experience was entirely dependent on how proactive the pilot was when communicating. Some pilots were able to get through the game without needing to use more than two thruster directions, leading to Engineers who were not very engaged.
5.0 Discussion and Recommendations

5.1 Discussion

Out of all 32 of our playtesters, 19 of them said that their favorite part of our prototype was learning how to master the movement systems or figuring out communicate well, the two most essential parts of our prototype. Based this statistic, we can say with confidence that our unusual cooperative gameplay is feasible, even with first-time players. The biggest limitation of our feasibility test was the small sample size, but we had a finite number of WPI students who could playtest our game, and Meow Wolf employees had trouble fitting our playtest into their busy schedules. Another factor limiting the number of playtesters we had was cooperative nature of our game. If we didn’t need two players for each playtesting session, it would have been much easier to playtest at Meow Wolf. Despite these limitations, our playtesters provided valuable feedback for how to improve our game.

Furthermore, playtesters enjoyed our unusual input method. Despite how Engineers did not have enough to do, they found the tactile feedback of the power cables to be very satisfying. Therefore, we can conclude that we were successful in incorporating an unconventional input method into our game.

5.2 Physical Recommendations

5.2.1 Pilot’s Physical Interface

We highly recommend using a joystick to control the ship’s thrusters, as it is an intuitive control scheme that fits the aesthetic of flying a spaceship. One concern that we have with the use of a joystick is that we observed a correlation between score and the direction of the set goal. When using a joystick as the Pilot’s controls it seems it is much easier to navigate the ship north. This could potentially be addressed by only generating the goal in the upward direction.

Another observation that we made of playtesters was that they had difficulty navigating the ship due to the binary nature of the joystick. Testers would try to more gently navigate the ship by slightly leaning the joystick believing this would cause the ship to turn, or move slower, but, while the joystick is physically capable of this, we had not programmed the joystick to perform that way. Our implementation simply checked if the joystick passed a certain threshold and gave that thruster full power. Therefore, we recommend implementing analog joystick inputs for the Pilot’s interface. This could be implemented by assigning two bits to each direction rather than just one. This would allow for three levels of thrust in each of the three directions represented by 01, 10, and 11. An example of the embedded code that could be used on the microcontroller is included in Appendix O.
5.2.2 Engineer’s Physical Interface

From our playtests we received positive feedback on the materiality of the Engineer’s plug-based interface, though 50% of Engineers said that they did not have enough to do. This can be solved by implementing more systems for the Engineer to manage, which we have already presented ideas for in our design document in Appendix D.

In the current state of our prototype the two interfaces are linked by a cable that powers the available systems lights on the Pilot’s side of the physical prototype. This cable is currently connected directly to both Arduinos in the two boxes and can be very inconvenient when moving or changing the prototype setup. We recommend making sure that any two separate interfaces are not connected by a permanent cable. It would make most sense to have connectors on either box that could be connected by a cable of varying length to allow for different setups, or disconnected entirely to allow for easier transportation. This would also allow for much easier maintenance.

5.2.3 Other Physical Recommendations

Playtesting revealed some insights about how truly difficult it is to coordinate things between two different people under the amount of time pressure our game provides. One concern that arose from this is that the original intended arcade for our game to be incorporated in might not be the optimal environment for it. In a busy arcade with many other players playing many other games, communication would be even harder due to the surrounding noise. Some groups struggled with communication even in the quieter setting of the Fort Marcy apartments or the Meow Wolf employee lounge. Therefore, we strongly recommend considering dedicating a full room-sized space to our game, as that would minimize the amount of external distractions to the players. A full room would also offer other advantages, such as allowing the game to be even more immersive, with larger displays for the pilot and more suitable lighting and decoration for the entire room. The engineer’s patch bays could even be spaced out across more of the room, making the engineer having to sprint between different stations to engage those systems.

5.3 Digital Recommendations

One of our strongest recommendations for the game would be to use an established game engine to develop the final product. While Processing was very effective to use while prototyping the game and rudimentary game mechanics, using a game engine would be significantly more efficient and would provide much more effective resources for mechanics such as collision detection. We recommend Unity as it has a very effective built-in physics engine (Unity, 2018), which will help a lot with making collisions feel right, given how much of a focus on physics our game has. Unity also interfaces with Arduino through the use of Uniduino, a plugin that allows Arduino to communicate with Unity (Uniduino, 2018).
The physical prototype was designed to integrate into our existing software, rather than replace it. Had we known we would be building the physical prototype from the start, we would have made significant changes. We would not have built a digital prototype interface for the Pilot, instead we would have had a single computer control both the Engineer’s screen and the game screen, with the Pilot’s controls being handled on an Arduino. The Engineer’s screen and the game screen would be displayed on separate monitors connected to the same computer. The power cables and the joystick would all be wired to a single Arduino. This would reduce the number of systems that can fail while the game is running, minimizing the risk of errors. This also completely removes the risk of network issues, which occurred during some of our playtests, rendering the game momentarily unresponsive.
6.0 Conclusion

We accomplished more than we originally set out to do. Originally, we were just going to create a design document, but our scope expanded significantly, and now we have successfully proven the principles of that design document through playtesting. Our game can fit in well either as a traditional arcade cabinet, to be incorporated into the arcade in the House of Eternal Return, or as a stand-alone experience, given a full room all to itself. Playtesting showed that our unconventional input of using plugs to enable game systems was viable, even with first time players. Overall, we achieved our goal of broadening the range of experiences available at Meow Wolf by creating a new kind of cooperative game experience for them.
References


Appendix

Appendix A: Pilot embedded code

```c
char count=0;
int sensitivity = 300; // Threshold to trigger direction update

void setup() {
    Serial.begin(9600); // Setup serial connection
}

void loop() {
    count=0;
    if(analogRead(0)>(1024-sensitivity)){ // Add 0001
        count+=0x01;
    }
    if(analogRead(5)>(1024-sensitivity)){ // Add 0010
        count+=0x02;
    }
    if(analogRead(0)<sensitivity){ // Add 0100
        count+=0x04;
    }
    if(analogRead(5)<sensitivity){ // Add 1000
        count+=0x08;
    }
    delay(10);
    Serial.print(count);
}
```
Appendix B: Engineer embedded code

```c
char count=0;

void setup() {
    Serial.begin(9600);//Setup serial connection
    //Setup pins as digital inputs
    pinMode(7,INPUT);
    pinMode(6,INPUT);
    pinMode(5,INPUT);
    pinMode(4,INPUT);
    //Setup pins as digital outputs
    pinMode(13,OUTPUT);
    pinMode(12,OUTPUT);
    pinMode(11,OUTPUT);
    pinMode(10,OUTPUT);
}

void loop() {
    count=0;
    //Reset pins at beginning of loop
    digitalWrite(13,LOW);
    digitalWrite(12,LOW);
    digitalWrite(11,LOW);
    digitalWrite(10,LOW);
    if(digitalRead(7)==HIGH){//Add 1000
        count+=0x08;
        digitalWrite(13,HIGH);
    }
    if(digitalRead(6)==HIGH){//Add 0100
        count+=0x04;
        digitalWrite(12,HIGH);
    }
    if(digitalRead(5)==HIGH){//Add 0010
        count+=0x02;
        digitalWrite(11,HIGH);
    }
    if(digitalRead(4)==HIGH){//Add 0001
        count+=0x01;
        digitalWrite(10,HIGH);
    }
```
count+=0x01;
digitalWrite(10,HIGH);
}
delay(10);
Serial.print(count);
Appendix C: Playtest Script

Interview Preamble:
Thank you for choosing to playtest our game and allowing us to listen to your feedback. Your feedback will be helpful for considering ways to improve the game in the future. Your participation in the playtest and interview is completely voluntary and all of your responses will be kept anonymous unless you give us permission to identify you by name. This interview will be recorded unless you do not wish this and the recording will later be reviewed for additional insight.

- Would both of you be okay if we recorded this?
- Would it be okay if we identify you by name in our report?

Demographics:
What is each of your prior experience with video games before?
What is your age?
What is your gender?

For Each Role:
Did you feel like you had enough to do as Pilot/Engineer?
What was the most frustrating part or aspect of what you just played?
What was your favorite part or aspect of what you just played?
If you had a magic wand and could change one aspect of the game, what would you change?
Do you have any final comments?

Asked Once:
What other name do you think would be appropriate?
The Starship Calamity

A Cooperative Adventure of Survival

Design Document

By:
Matthew Grim
Griffen Spincken
Nicolas Pingal
Overview

Narrative

Themes

Background Story

Gameplay Mechanics

Levels:

Goal/Objectives

Game Levels:

Core Mechanics

Overview

Power Cables

Pilot

Engineer

Power Management

Ship Movement

Asteroids

Special Asteroids

Pirates

Hull Integrity

Momentum

Pilot

Overview

Controls

Controls

Thrusters

Special Systems

Emergency Stop

Gravity Pulse
Repair Drone
Emergency Battery
Passive Abilities
Cameras
Engineer
Overview
Controls
Radar
Aesthetic
Cabinet Design
Dimensions
In-Game Graphics and Art Style
Sound
Physical Components
Hardware Systems
Communication Scheme
Embedded systems
Pilot Interface Components
Joystick
Buttons
Screen
Engineer Interface Components
Power Cables
Plug
Cable
Receptacle
Gauges
Power Gauge
Hull Integrity
Radar
Recommendations

Suns
Room Scale
Alternative Pilot Control Schemes
Alternate Engineer Radar Systems
Shooting Mechanics
Combination Systems
Power Levels
Overview

Starship Calamity is a game where two players cooperate to navigate a damaged, barely-functioning spaceship through hostile space to accomplish a range of goals for their corporate overlords. Each of the two players has a unique role with exclusive functions: the Pilot has to navigate the ship through space, focusing on more action-based gameplay, while the Engineer has to manage the ship’s power and the Pilot’s available functions, focusing on slower gameplay. Only through careful communication and cooperation can both players succeed.

Narrative

Themes

The main aesthetics are fellowship and challenge. The aesthetics of challenge comes from the nature of the game as a series of challenges, focused less on narrative, and more on challenge for challenge’s sake. The aesthetic of fellowship comes from the cooperative nature of our game.

Background Story

The players’ ship is the Starship Calamity, a horrifyingly old and beat up transport frigate, owned by Space Corp, used to transport rare minerals mined from asteroids on the far reaches of human-explored space. The job of the day is fairly mundane, until an accidental collision with an asteroid impairs the ship’s power systems, and attracts the attention of space pirates in the area.

Gameplay Mechanics

Levels:
The game is divided into levels, each with its own set of basic objectives. These range from picking up and delivering materials from one planet to another, searching for a wormhole to escape an asteroid field, or landing on a specific base on a specific asteroid.

Goal/Objectives

The goal of the game, is to find and complete the level’s objective. These can be broken down into one or a combination of these types of objectives: search, deliver, race/time trial, or survive.

Deliver: The ship must first locate and pick up a package, then deliver that package to a specific location.

Race/Time Trial: The ship must navigate its way through a series of objectives before time runs out, or before the enemy can do so. This works as a standalone game mode, however it could be used with others for longer levels.
Land on: The team must navigate the ship to a specific objective, such as a planet, and land on it. If the ship is going too fast, then it will explode and the mission will fail. In this mode the asteroids will not move, to make navigation and landing on goal easier, because if they moved, they may hit you and force you off the objective.

Survive: Survive as long as possible against the enemies and environment, or until time runs out.

Game Levels:
The levels are free-roaming and 2D, populated by asteroids, objectives, and enemies. The players view the arena from a top-down perspective. See below for descriptions of asteroids, suns and enemies. The game arena is procedurally generated, to allow for endless game arenas. This involves the game generating a semi-random environment rather than a manually constructed one. Objectives have an outer bounds to their spawn location, so they cannot be infinitely far away from the ship’s spawn point, if the level has an objective that requires a bounds.

Core Mechanics

Overview
The game is designed for two players, and each assumes a different role with different interfaces. One player is the Pilot, and the other is the Engineer. The Pilot is in control of the ship and can use the array of controls to maneuver the ship, however not all controls can be active at once. The Engineer is able to choose which controls are active at any given time by plugging and unplugging “power” cables. The Pilot is the only player who can see the game screen. The Engineer has a radar that can see the locations of in-game objects, but not what those objects are.

Power Cables
The ship has a limited number of power cables. These are used to power the Pilot’s controls. The Engineer decides where they are plugged in. There should be power cables half of the total systems of the ship, rounded down. The power cables are to be physically separated from the Pilot controls so one person cannot fill both roles at once.

Pilot
The Pilot is capable of steering the ship and viewing its immediate area. The Pilot is in charge of navigating the ship past the various hazards and getting the ship as far as possible. The Pilot’s control scheme includes a joystick, for using the thrusters, three buttons, for using the Special Systems, and the Emergency Battery button.

Engineer
The Engineer’s role is to manage power cables, communicate the locations of incoming asteroids on the radar, and keep the Pilot informed about the status of power and hull integrity
meters. These meters are hull integrity (health) and power. The Engineer can also see the distance to the objective, but not the direction it is in.

**Power Management**
The ship has a finite amount of power, which is only able to be restored by entering orbit around suns or finding “power pickups”. As such, the two players need to carefully coordinate power use by only engaging and using systems that are absolutely necessary. Thruster systems consume power when engaged and activated, special systems consume power to charge up, and passive systems do not consume power at all, however must be plugged in to activate. The ship starts to lose “health” quickly after all power has been consumed. Only the Engineer can view the remaining power level.

**Ship Movement**
The ship’s momentum is conserved, with only very small amounts of drag to slow the ship. The ship’s direction can be manipulated through the use of four thrusters. The rear thruster controls forward movement, and is more powerful than the brake thruster, used to slow down. Two side thrusters exist for rotation and are required for directional changes.

**Asteroids**
The asteroids are the primary obstacle of the game. They come in varying sizes. If the ship hits one, it will take damage depending on the size of the asteroid, with larger asteroids dealing more damage. Asteroids will spawn off screen and, depending on the level being played, will either move towards the area shown on screen, or will remain stationary. Asteroids will be destroyed after an impact with the ship.

**Special Asteroids**
Special asteroids will spawn in the same manner normal asteroids will. These special asteroids act the same as normal asteroids, except that if they deal damage to the ship, they also disable one of the power cables for a short time, reducing the number of systems that can be engaged at once.

**Pirates**
Pirates will pursue the player ship throughout the game area during certain levels. They follow the same spawn rules as asteroids. They are capable of turning and intelligent movement in order to pursue the player. They damage the player by ramming into the ship. They do not shoot at the ship because they want to board and loot the ship instead of just destroying it. They deal a large amount of damage when they hit and board the ship.

**Hull Integrity**
Hull integrity is measured as a percentage. It is reduced by hitting asteroids or being hit by enemies. Should it hit 0% then the ship explodes and the game ends. It can be repaired through use of certain abilities.
Momentum
Because the ship is in space, carefully managing momentum is key, as there is little drag to slow the ship. Furthermore, the rear thruster is more powerful than the brake thruster, so slowing can be a challenge.

Pilot
Overview
The Pilot's job is to navigate the ship through the myriad obstacles in space. Compared to the Engineer's role, the Pilot has to manage more spatial information regarding the area around the ship and the safest path forward.

Controls
Figure A shows the Pilot's interface. The Pilot's control panel is composed of a joystick used to control the four thrusters, arranged on top of a diagram of the ship. Three smaller buttons are used to activate the ship's three special systems when charged (see Special Systems under Controls). They begin dark, slowly blink while charging, and remain continuously illuminated when fully charged. The big red button is used to activate the emergency battery.

Controls
Thrusters
There are 4 thrusters on the ship. One rear thruster used as the main propulsion system, left and right thrusters used for rotation, and one front thruster used to brake. The rear thruster is fifty percent more powerful than the brake thruster, necessitating careful turning. These thrusters are activated with the joystick, which allows for up to two of them to be activated at once.

Special Systems
The ship has a range of special systems that need to be engaged, and must be charged up in advance before they can be used. These temporarily expand the ship's capabilities. Unlike thrusters, once they are used, they deactivate and must be fully charged to be used again.

Emergency Stop
This ability immediately stops the ship, removing all current momentum.
**Gravity Pulse**
A rechargeable pulse used to push asteroids out of the way of the ship. It projects out just in front of the ship.

**Repair Drone**
A repair drone used to repair damage to the ship’s hull integrity. This system does not affect anything outside the ship.

**Emergency Battery**
The emergency battery is a unique system that may only be activated once per game. It engages every system on the ship, and fully charges every special system, for a few seconds. Systems recharge much faster during this time, allowing them to be used more often.

**Passive Abilities**
These systems are screens that show important information to both the Pilot and the Engineer. They do not consume power when engaged, but just require a power cable.

**Cameras**
The cameras affect what the Pilot can see. If the camera system is engaged, the Pilot will have a full view of the game screen. When the camera system is not engaged, then the view will change to a small cone of vision directly in front of the ship, as if the Pilot can only see out the windows.

**Engineer**

**Overview**
The Engineer’s primary job is the management of power, which enables the Pilot to navigate the ship. The Engineer cannot directly control the ship at all or see the nature of the various threats approaching the ship, but can enable or disable all the ship’s systems as they are needed.
Controls
Figure B shows the Engineer’s interface. The Engineer interacts with the game by plugging power cables into the patch bays to enable the ship’s systems. The four patch bays on the black square control the thrusters, the three with the LED rings control the emergency stop, gravity pulse, and repair drone special systems, and the two rightmost patch bays control the Engineer’s radar and Pilot’s screen. The distance to the objective is displayed as well, if that game mode has an objective. The Engineer’s interface also includes two meters, one for power and another for hull integrity.

Radar
The radar, shown on the far right of the upper panel, gives direction and distance of asteroids and enemies that are out of the Pilot’s visible range, displaying these objects as just green dots. It does not give indication of what the object is. The radar also displays how far the Pilot is from the objective, but not what direction it is in.

Aesthetic

Cabinet Design
Dimensions
70-74” height, 28-32” wide, 32-34” deep, ~32” height for control panel. Control Panel: Width of cab. ± 0-4” on each side, 12-14” deep, 4-6” height, can slope up or stay flat. Usually slopes up to a +2-3” other end. Monitor to fit.
These are dimensions the ranges found after examination of arcade cabinet schematics.

In-Game Graphics and Art Style
The in-game graphics are drawn in a pixel-art style, to fit in better with the arcade setting.
Sound
The sounds for this game should be in line with the visual aesthetic. Ideally they should be as if they were from an old space movie. Lots of theremin, make it sound as stereotypically space sounding as possible.
Physical Components

Hardware Systems

Communication Scheme
Communication between the embedded system and the game system will be handled using a serial USB connection.

Embedded systems
Control inputs can be handled through the use of microcontrollers. For this we recommend the use of Arduino or Teensy chips due to their availability, low cost, and ample documentation. These boards will handle inputs from both of the players interfaces and will send this information to the game system across as USB connection.

Pilot Interface Components

Joystick
The Pilot’s control panel includes a joystick to control various systems such as thrusters on the ship. The joystick should be 8 directional, so the Pilot can fire two thrusters at once if he needs to.

Buttons
Standard buttons. They come back up after being pushed down. Arcade styles button are preferable for their durability. See picture in Pilot Controls section for examples of arcade button.

Screen
The Pilot’s visual interface will be displaced an LCD screen connected to the main game console via a HDMI connection.

Engineer Interface Components

Power Cables

Plug
This will be a durable piece of plastic or wood that houses the magnet that will both hold it in the receptacle as well as trigger the Hall Effect sensor. The plug itself is cylinder one
inch in diameter with a 1.5 inch diameter grip. The grip and plug will be attached using an epoxy or similar strong adhesive that will survive repeated use. We decided to have the cable carry a signal as we found that a magnet-based switch would be much more durable and less prone to failure. In this design the presence of the magnetic plug is sensed by its receptacle.

**Cable**
This will be a clear plastic hose with individually addressable led strips contained within them. The hose will then be filled with clear silicone both to diffuse the light from the LEDs and to provide the necessary durability to survive public use. Using strips such as the Adafruit Neopixel strips we can provide visual feedback to the player using changing colors and one-dimensional animations along the cable itself.

**Receptacle**
Each power receptacle will be made from durable plastic that will snugly fit the power plug. The back of the receptacle will have a steel washer that the magnet in the plug will attract to. This will hold the plug in place in addition to the physical coupling of the plug. The center of the washer will house a hall effect sensor. This is a variable resistance sensor that detects the strength of magnetic fields. These sensors will report be connected to an Arduino Uno that will process what plugs are currently connected and report this via bitstream over a serial connection to the system that the game will be running on.

**Gauges**
Power Gauge

Total system is represented by a bar graph of RGB LEDs such as Adafruit’s line of NeoPixel strips. These can be controlled by an Arduino Uno that would control the display for both the power gauge and the hull integrity. The System would receive information over a serial connection from the main game system. This Arduino would then convert the bitstream to a percentage and would represent this using the LED strip as a bar graph.

Hull Integrity

Hull integrity represents the remaining health that the players have until they reach a game over. This will be represented by three seven segment displays to numerically display this percentage in decimal. This can be controlled by an Arduino Uno that would control the display for both the power gauge and the hull integrity. The System would receive information over a serial connection from the main game system. This Arduino would then convert the bitstream to a percentage and would represent this using the LED segment displays.

Radar

The Engineer’s radar view would be displayed using a small LCD screen that be connected to the main game system through a HDMI connection.
Recommendations

Suns
Suns are a potential power replenishment system in the game. The ship would regain power when it is in close proximity to a sun. The power replenished to the ship is dependent on the distance to the sun. There is a specific end zone where the sun will no longer give you power, so as to avoid ambient power absorption. Gaining power should be a dedicated task. Should the ship crash into a sun, it would explode and the game will end.

Room Scale
Instead of making a double-sided arcade cabinet, the game would be in a room of its own, split down the middle for the Engineer and Pilot sides. The display would be much larger and more immersive. After playtesting showed how difficult communication was even in a quieter environment, we highly recommend considering a room-scale setup for this game.

Alternative Pilot Control Schemes
Consider using other means of controlling the ship on the Pilot's end, such as a joystick or a pair of joysticks, where one only moves up and down, and one only controls the turning. Possibly even something like a control yoke or a plane style cockpit.

Alternate Engineer Radar Systems
Consider giving the Engineer more or less information on the radar under certain conditions. For instance, the radar could display the direction of the objective, but only when the ship is at a complete stop.

Shooting Mechanics
A common request from our playtesters was the ability to shoot the asteroids or interact with them in some more meaningful way. This could be incorporated into the game in a number of ways: there could potentially be a dedicated third “Gunner” role, who would have to coordinate with the engineer as well as the pilot. Alternatively, the Pilot could controls the guns themselves, though that would add potentially too much more for the pilot to do.

Combination Systems
Another request brought up by playtesters was the ability to use secret moves by plugging in specific combinations of systems. For instance, activating a turbo thruster by engaging all of the thrusters at once, or using one of the special abilities with the thrusters for a unique effect. This could add another layer of strategy to the Engineer’s gameplay.
**Power Levels**

One idea brought up during playtesting was the ability for the Engineer to control the amount of power going to each system when they are engaged. For instance, the Engineer could raise or lower the amount of power going to each thruster, which would make them more or less powerful, respectively. This would give the Engineer significantly more control over the ship, so we highly recommend considering this.
Appendix E: Joystick Specifications Sheet

SPECIFICATIONS

**ELECTRICAL**
- SUPPLY VOLTAGE: 5.9V DC
- CENTER VOLTAGE: 2.5V DC
- SUPPLY CURRENT: 10mA max (axis)
- OUTPUT CURRENT: 2mA max (axis)
- EFFECTIVE OUTPUT: 0V – 5V DC
- OUTPUT TOLERANCE: ±2%

**MECHANICAL**
- OPERATING TRAVEL CENTERING: 40° (20° from center)
- BREAKOUT FORCE: Single spring, omni-directional
  - OPERATING FORCE: 0.663N
  - MAXIMUM FORCE: 1.305N
- OPERATING TEMP: -40°C to +85°C
- LIFE EXPECTANCY: 10 million operations

**WIRING**
- RED: Supply voltage
- BLACK: Ground
- BLUE: X-Axis
- YELLOW: Y-Axis
- ORANGE: Switch 1
- VIOLET: Switch 2
- WHITE: Switch common

RoHS compliant

**Note:** Actual strain relief position may vary from specifications.

**CH Products**

**Series III – 2 Axes, Stock Grip, 2 Top Buttons**

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED HEREIN IS THE PROPRIETARY PROPERTY OF CH PRODUCTS AND IS PROVIDED FOR INTERNAL USE ONLY. REPRODUCTION OR DISCLOSURE TO OTHERS WITHOUT THE WRITTEN PERMISSION OF CH PRODUCTS IS PROHIBITED.
Appendix F: Engineer Circuit Diagram
Appendix G: Pilot LED Circuit Diagram
Appendix H: Pilot Joystick Circuit Diagram
Appendix I: Color Surfing Design Sketches

Display Option #1

Display Option #2

Current color mix

LCD Screen

Off LED represents player location

Rest of strip is current color

RGB LED Strips

Strips possibly under diffusing film
Appendix J: Photos of Physical Interface
Appendix K: Screen Code for Starship Calamity

import java.awt.Color;
import java.awt.Point;
import java.util.ArrayList;
import oscP5.*;
import netP5.*;
import processing.serial.*;

//Game variables
boolean fade = true;
int count;
int difficulty = 5; // 0-100
int score;
Serial joy;
boolean xInvert = false;
boolean yInvert = false;
Point end = new Point(0, 0);
boolean win = false;
double distance;
double initDistance;
boolean noPower = false;
int respawn = 0;
float initTime;
float tim;

//Sound variables
//SoundFile rocket;
//SoundFile crash;

//Communication variables
OscP5 oscP5;
NetAddress listener;
String engiIP = new String("192.168.137.99");

//Ship sprites variables
PImage ship0;
PImage ship1;
PImage ship2;
PImage ship3;
PImage ship4;
PImage ship5;
PImage ship6;
PImage ship7;

//Asteroid variables
int astMin = 40;
int astMax = 120;
ArrayList<Asteroid> asteroids = new ArrayList<Asteroid>();
int asteroidDensity = 35; // field density
int astBuf = 2000; // asteroid buffer
int astDen = 12; // asteroid mass density
int astDespawn = 2000; // Despawn Radius
int astMom = 0;

// Star variables
ArrayList<Point> stars = new ArrayList<Point>();
int starDensity = 18000;
int twinkle = 150; // Star twinkle (0-255)
int buf = 5000; // star buffer
int starSize = 3;

// Ship variables
Momentum shipM = new Momentum(200, 0, 0.1);
Rmomentum shipRm = new Rmomentum(200);
int speedLimit = 15;
double rotLimit = 0.25;
int thrust = 30;
int rotThrust = 30;
double rotDrag = 0.3;
double drag = 80;
int health = 255;
int shipBox = 45;
boolean alive = true;
int mainEngineFactor = 2;
int xPos;
int yPos;
double shipR;
int deltaX;
int deltaY;

// Debugging tools
boolean hitBox = false;
boolean healthBar = false;
boolean recHealthBar = false;
boolean controlsConsole = false;
boolean framerateConsole = true;
boolean stopCheat = true;
boolean killCheat = true;
boolean healCheat = true;
boolean joystick = false;

// Controls variables
boolean up;
boolean down;
boolean left;
boolean right;
boolean eBrake;
void setup() {  
    //size(1000, 1000);
    fullScreen();

    noStroke();
    smooth();
    //load ship sprites
    ship0 = loadImage("newships/newnone.png");
    ship1 = loadImage("newships/newb.png");
    ship2 = loadImage("newships/newl.png");
    ship3 = loadImage("newships/newr.png");
    ship4 = loadImage("newships/newlr.png");
    ship5 = loadImage("newships/newlb.png");
    ship6 = loadImage("newships/newrb.png");
    ship7 = loadImage("newships/newall.png");

    if(joystick)//Set up joystick controls
        joy = new Serial(this, Serial.list()[0], 9600);

    //defaults screen position to center
    xPos=width/2;
    yPos=height/2;

    starsPop();//populate star arrayList
    astPop();//populate meteor arrayList
    goal();//Generate end goal

    //setup for network communications
    oscP5 = new OscP5(this, 4444);
    oscP5.plug(this, "piloting", "/pilot");
    oscP5.plug(this, "ing", "/engi");
    listener = new NetAddress(engiIP, 4444);

    frameRate(30);
    imageMode(CENTER);
    ellipseMode(CENTER);
    reset();
}

void draw() {

    //This state is the main game loop
    if(alive){
        distance = dist((float)end.getX(), (float)end.getY(), 0, 0);
background(0);//Draw background
starWrap();//Wrap stars if they leave the buffer zone
pushMatrix();
drawStars();//Draw stars on background
handleAsteroids();//Handle asteroids
popMatrix();//Pops stars and asteroids

//Draw rectangular health bar
if(recHealthBar){
    fill(255-health,health,0,120);
    rect(40,40,1.5*health,40);
}

//Draw goal
fill(153,50,204,200);
ellipse((float)end.getX()+width/2,(float)end.getY()+height/2,300,300);

//Move ship based on momentum
xPos+=(int)shipM.getX();
yPos += (int)shipM.getY();
deltaX=xPos-width/2;
deltaY=yPos-height/2;

drawShip();//Draws the ship
selectSprite();//Selects what sprite to draw
playerInputs();//Check player inputs

if(health<=0){//Checks to see if player is dead
    alive=false;
    win = false;
    count=0;
}
else if(distance<200){ //Checks to see if player has won
    win=true;
    tim = millis()-initTime;
    alive=false;
    count =0;
}
else if(!win && fade && count++ < 150){//Win fade state
    asteroids.clear();
    fill(255,255,255,20);
    rect(0,0,width,height);
}
else if(win && fade && count++ < 200){//Win fade state
asteroids.clear();
fill(0,0,0,20);
rect(0,0,width,height);
}
else if(win){
    win();//Win screen
}
else{
    lose();//This state asks the player if they wish to try again
}
sendToEngi(health,distance,respawn);//Send health,distance, and a flag for respawns

//Debugging tools
shipBox(hitBox);//Draw ship hitbox
drawHealth(healthBar);//Draw HeathBar
printControls(controlsConsole);//Print Pilot inputs to console
printInfo(frameCount, framerateConsole);//Print framerate and information to console every 60 frames
}

//Momentum object
class Momentum{
    double xCom;//X component of velocity
    double yCom;//Y component of velocity
    double mass;//Mass of object

    //Constructor
    public Momentum(double inMass){
        xCom=0;
        yCom=0;
        mass=inMass;
    }

    //Constructor with initial momentum
    public Momentum(double inMass,double x,double y){
        xCom=x;
        yCom=y;
        mass=inMass;
    }

    //Returns X component of velocity vector
    public double getX(){
        return xCom;
    }

    //Returns Y component of velocity vector
    public double getY(){

return yCom;
}

//Returns mass of object
public double getMass(){
    return mass;
}

//Applies a force to the object to accelerate it
public void applyForce(double xForce, double yForce){
    xCom+=xForce/mass;
    yCom+=yForce/mass;

    if(xCom>speedLimit){
        xCom=speedLimit;
    }
    if(yCom>speedLimit){
        yCom=speedLimit;
    }
    if(xCom<-speedLimit){
        xCom=-speedLimit;
    }
    if(yCom<-speedLimit){
        yCom=-speedLimit;
    }
}

//Stops all movement
public void eStop(){
    xCom=0;
    yCom=0;
}

//Applies a drag force opposite to the momentum's movement and returns the x and y components of this drag vector as a Point object
public Point decay(double force){
    if(xCom==0&&yCom==0){
        return new Point(0,0);
    }
    this.applyForce(-xCom*force/100,-yCom*force/100);
    return new Point((int)(-xCom*force/100),(int)(-yCom*force/100));
}
}

//Rotational momentum object definition
class Rmomentum{
    double mom; //Rotational momentum magnitude
    double mass; //mass of object
// Constructor
public Rmomentum(double inMass) {
    mom = 0;
    mass = inMass;
}

// Returns magnitude of momentum
public double getMom() {
    return mom;
}

// Returns mass of object
public double getMass() {
    return mass;
}

// Applies a force to rotate the object
public void applyForce(double force) {
    // F = MA
    mom += force / (mass * 100);

    if (mom > rotLimit) {
        mom = rotLimit;
    } else if (mom < -rotLimit) {
        mom = -rotLimit;
    }
}

// Stops all rotation
public void stop() {
    mom = 0;
}

// Applies drag to rotation
public void decay(double force) {
    if (mom == 0) {
        return;
    } else if (mom > -0.000001 && mom < 0.000001) {
        this.stop();
    } else if (mom > 0) {
        mom -= force / 1000;
    } else if (mom < 0) {
        mom += force / 1000;
    }
}
//Asteroid object definition
class Asteroid{
    private Momentum mom;//Momentum for asteroid
    private Rmomentum rmom;//Rotational momentum for asteroid
    private int size;// Size of asteroid
    private Point loc;//Location of asteroid
    private Color col;

    //Basic Constructor
    public Asteroid(int inSize, int x, int y){
        loc = new Point(x,y);
        size = inSize;
        mom = new Momentum(inSize*astDen);
        mom = new Momentum(inSize*astDen);
    }

    //Constructor with momentums
    public Asteroid(int inSize, int x, int y, Momentum inMom, Rmomentum inRot,Color inCol){
        loc = new Point(x,y);
        size = inSize;
        mom=inMom;
        rmom=inRot;
        col=inCol;
    }

    //move asteroid by dx and dy
    public void translate(int dx, int dy){
        loc.translate(dx,dy);
    }

    //Return asteroid x coordinate
    public int getX(){
        return (int)loc.getX();
    }

    public Color getCol(){
        return col;
    }

    //Return asteroid x coordinate
    public int getY(){
        return (int)loc.getY();
    }

    //Return size of asteroid
public int getSize(){
    return size;
}

//Return momentum object for asteroid
public Momentum getMom(){
    return mom;
}

//Return rotational momentum object for asteroid
public Rmomentum getRmom(){
    return rmom;
}

//sets asteroid location to that of Point object
public void moveTo(Point newloc){
    loc = newloc;
}
}

void playerInputs(){
    if(joystick){
        while(joy.available() > 0) {
            char inByte = joy.readChar();
            //println(inByte);
            if((inByte&0x08)==0x08){//Up
                up = true;
            }else{
                up = false;
            }
            if((inByte&0x04)==0x04){//Right
                right = true;
            }else{
                right = false;
            }
            if((inByte&0x02)==0x02){//Back
                down = true;
            }else{
                down = false;
            }
            if((inByte&0x01)==0x01){//Left
                left = true;
            }else{
                left = false;
            }
        }
    }
}
left = false;
}

//Buttons
if((inByte&0x10)==0x10){ //Button 1
eBrake = true;
}

}
}
invert(xInvert,yInvert);

//Apply force to ship main or rear thrusters
if((keyPressed&&key=='s')||down){

shipM.applyForce(thrust*cos((float)(shipR+PI/2)),thrust*sin((float)(shipR+PI/2)));
  //if(!rocket.isPlaying()){%
  //  rocket.play();%
  //}
}

if((keyPressed&&key=='w')||up){

  shipM.applyForce(-thrust*mainEngineFactor*cos((float)(shipR+PI/2)),-
thrust*mainEngineFactor*sin((float)(shipR+PI/2)));
  //if(!rocket.isPlaying()){%
  //  rocket.play();%
  //}
}

//Apply torque to ship with side thrusters
if((keyPressed&&key=='a')||left){
  //if(!rocket.isPlaying()){%
  //  rocket.play();%
  //}
  shipRm.applyForce(-rotThrust);
}
if((keyPressed&&key=='d')||right){
  //if(!rocket.isPlaying()){%
  //  rocket.play();%
  //}
  shipRm.applyForce(rotThrust);
}

//if(rocket.isPlaying()&&!up&&!down&&!left&&!right){%
  //  rocket.stop();%
  //}
//Cheat controls
if(eBrake||(key=='b'&&stopCheat&&keyPressed)){//Full eStop
    shipM.eStop();
    shipRm.eStop();
eBrake=false;
}
if(key=='k'&&killCheat&&keyPressed){//Kill ship
    health-=10;
}
if(key=='h'&&killCheat&&keyPressed){ //Heal ship
    health+=1;
}

void translateStuff(){
    //Translate stars based on ship movement
    for(int i=0;i<stars.size();i++)//scroll stars
        stars.get(i).translate(-1*(int)shipM.getX(),-1*(int)shipM.getY());
    //Translate asteroids based on ship movement
    for(int i=0;i<asteroids.size();i++)//scroll asteroid
    {
        int dx=0;
        int dy=0;
        dx=(int)asteroids.get(i).getMom().getX();
        dy=(int)asteroids.get(i).getMom().getY();
        asteroids.get(i).translate(-1*(int)shipM.getX()+dx,-1*(int)shipM.getY()+dy);
    }
    end.translate(-1*(int)shipM.getX(),-1*(int)shipM.getY());
}

void handleAsteroids(){
    for(int i=0;i<asteroids.size();i++){
        Color col = asteroids.get(i).getCol();
        fill(col.getRed(),col.getGreen(),col.getBlue());
        ellipse(asteroids.get(i).getX(),asteroids.get(i).getY(),asteroids.get(i).getSize(),asteroids.get(i).getSize());
        //COLLISION DETECTION
        if(dist(asteroids.get(i).getX(),asteroids.get(i).getY(),width/2+ship0.width/2,height/2+ship0.height/2)<asteroids.get(i).getSize()/2+shipBox/2){
        }
    }
```java
//println("Collision");
double impact =
astDen*asteroids.get(i).getSize()*sqrt((pow((float)shipM.getX(),2)+pow((float
)shipM.getY(),2)));
Point impulse = shipM.decay(impact);
fill(255,0,0,pow((float)impact/2000,2)+80);
rect(0,0,width,height);
health-=pow((float)impact/2000,2);
//fill(169,104,54);
asteroids.get(i).getMom().applyForce(-impulse.getX(),-impulse.getY());
//crash.play();
}

//Kill asteroids as they leave the buffer
if((asteroids.get(i).getX()>astDespawn)||(asteroids.get(i).getY()>astDespawn)
||(asteroids.get(i).getX()<astDespawn)||(asteroids.get(i).getY()<-
astDespawn))
    asteroids.remove(i--);//Kill asteroid

    //Spawn new asteroids to replace the killed one
    float ran = random(100);
    int size = (int)random(astMin,astMax);
    if(ran<25){//Top
        asteroids.add(new Asteroid(size,(int)random(-
        (astBuf),width+(astBuf)),(int)random(-astBuf,-astMax),new
        Momentum(100,random(-astMom,astMom),random(-astMom,astMom)),null,astCol()));
    }
    else if(ran<50){//Bottom
        asteroids.add(new Asteroid(size,(int)random(-
        (astBuf),width+(astBuf)),(int)random(astMax+height,height+(astBuf)),new
        Momentum(100,random(-astMom,astMom),random(-astMom,astMom)),null,astCol()));
    }
    else if(ran<75){//Left
        asteroids.add(new Asteroid(size,(int)random(-astBuf),-
        astMax),(int)random(-astBuf),height+(astBuf)),new
        Momentum(100,random(-astMom,astMom),random(-astMom,astMom)),null,astCol()));
    }
    else if (ran<100){//Right
        asteroids.add(new
        Asteroid(size,(int)random(astMax+width,astBuf+width),(int)random(-
        (astBuf),height+(astBuf)),new Momentum(100,random(-astMom,astMom),random(-
        astMom,astMom)),null,astCol()));
    }
    //spawn double on difficulty linked percentage
    if(random(100)<difficulty){
        ran = random(100);
        size = (int)random(astMin,astMax);
```
void selectSprite(){
    if((left&&right&&up&&down)||(up&&down)){
        image(ship7, 0, 0);
    }
    else if((left&&right)||down){
        image(ship4, 0, 0);
    }
    else if(right&&up){
        image(ship5, 0, 0);
    }
    else if(left&&up){
        image(ship6, 0, 0);
    }
    else if(up){
        image(ship1, 0, 0);
    }
    else if(left){
        image(ship3, 0, 0);
    }
    else if(right){
}
image(ship2, 0, 0);
}
else{
    image(ship0, 0, 0);
}
}

void starWrap(){
    for(int i=0;i<stars.size();i++){
        if((stars.get(i).getX()>width+buf)){
            stars.get(i).setLocation(-buf,stars.get(i).getY());
        }
        if((stars.get(i).getX()<buf)){
            stars.get(i).setLocation(width+buf,stars.get(i).getY());
        }
        if((stars.get(i).getY()>height+buf)){
            stars.get(i).setLocation(stars.get(i).getX(),-buf);
        }
        if((stars.get(i).getY()<buf)){
            stars.get(i).setLocation(stars.get(i).getX(),height+buf);
        }
    }
}

void drawStars(){
    for(int i=0;i<stars.size();i++){
        fill(255);
        //fill(255-random(twinkle),255-random(twinkle),255-random(twinkle));
        rect((int)stars.get(i).getX(),(int)stars.get(i).getY(),starSize,starSize);
    }
}

//populate screen with initial asteroids
void astPop(){
    for(int i =0;i<asteroidDensity;i++){
        int x = width/2;
        int y = height/2;

        while(dist(width/2,height/2,x,y)<100){
            x = (int)random(-astBuf,width+astBuf);
            y = (int)random(-astBuf,height+astBuf);
        }
        asteroids.add(new Asteroid((int)random(astMin,astMax),x,y,new Momentum(100,random(-astMom,astMom),random(-astMom,astMom)),null,astCol()));
    }
}
void goal() { // generates end goal for game
    while ((dist((float) end.getX(), (float) end.getY(), width/2, height/2) < 10000) || (dist((float) end.getX(), (float) end.getY(), width/2, height/2) > 15000)) {
        end = new Point(round(random(-15000, 0)), round(random(-15000, 15000)));
    }
    initDistance = dist((float) end.getX(), (float) end.getY(), 0, 0);
}

// Takes in integer and returns true if nonzero and false otherwise
public boolean intToBool(int in) {
    if (in != 0)
        return true;
    else
        return false;
}

// Send current Health to engineer
void sendToEngi(int in, double dist, int alive) {
    OscMessage m = new OscMessage("/screen");
    m.add(in);
    m.add(dist);
    m.add(alive);
    oscP5.send(m, listener);
    respawn = 0;
}

// Recieve Network Controls
void piloting(int inUp, int inDown, int inLeft, int inRight) {
    // println("Recieved Controls");
    up = intToBool(inUp);
    down = intToBool(inDown);
    left = intToBool(inLeft);
    right = intToBool(inRight);
}

// Recieve ship power from engineer
void engineering(int power) {
    if (!intToBool(power)) {
        health = 1;
        noPower = true;
    }
}

void printInfo(int frames, boolean print) {
    if (frames%60 == 0 && print){
println("FPS: "+round(frameRate)+", Health: "+health+", Distance: "
+round((float)distance)+", Pos: ("+deltaX+","+deltaY+
"), Vel: ("+floor((float)shipM.getX())+","+floor((float)shipM.getY())+"),
Ast: "+asteroids.size()+" Goal: "+end.getX()+", "+end.getY());
}
}
void printControls(boolean print){
  if(print){
    println("Up: "+up+, Down: "+down+, Left: "+left+, Right: "+right);
  }
}
void reset(){
  initTime=millis();
  alive = true;
  health = 255;
  count=0;
  score = 0;
  goal();
  xPos=width/2;
  yPos=height/2;
  noPower = false;
  respawn = 1;

  astPop();
  shipR=0;
  shipM.eStop();
  shipRm.eStop();
}
```java
fill(0,255,0);
ellipse(0,0,shipBox,shipBox);
}
}

void drawHealth(boolean draw){
    //Draw shipbox
    if(draw){
        fill(0,255,0);
        ellipse(0,0,shipBox,shipBox);
    }
}

void invert(boolean x, boolean y){
    //Invert x axis
    if(x){
        boolean hold = left;
        left=right;
        right=hold;
    }
    //Invert y axis
    if(y){
        boolean hold = up;
        up=down;
        down=hold;
    }
}

void lose(){
    background(150);
    textSize(32);
    fill(0);
    text("You made it: "+round(100*(float)((initDistance-distance)/initDistance))+'/'+"% of the way\n\nWould you like to try again?\n\nY/N"+
"\n\n"+round((tim)/1000),width/3,height/2);
    //Check for player input
    if(key=='y'){
        reset();//Reset game state
    }
    if(key=='n'){//Exits app
        //rocket.stop();
        //crash.stop();
        exit();
    }
}
```
void win()
{
    background(0);
    textSize(32);
    fill(255);
    text("Congratulations,
    \n    You made it in "+round((tim)/1000) +" seconds
    \n    Would you like to play again?
    \n    Y/N",width/3,height/2);

    //Check for player input
    if(keyPressed&&key=='y'){
        reset(); //Restart game loop
    }
    if(keyPressed&&key=='n') //Exits app
    {
        //rocket.stop();
        //crash.stop();
        exit();
    }
}

void starsPop()
{
    //populate star arrayList
    for(int i =0;i<starDensity;i++)
    {
        stars.add(new Point((int)random(-buf,width+buf),(int)random(-buf,height+buf)));
    }
}

Color astCol()
{
    return new Color((int)random(128,139),(int)random(0,82),(int)random(0,45));
}
import oscP5.*;
import netP5.*;
import processing.serial.*;

// flag for enabling keyboard engineer overrides
boolean wasdEngi = false;

// enables joystick input through arduino
boolean joystick = true;
Serial joy;

// keyboard inputs on the Pilot side
int upb = 0;
int downb = 0;
int leftb = 0;
int rightb = 0;

// keyboard inputs to transmit to the engineer and screen
int Pup = 0;
int Pdown = 0;
int Pleft = 0;
int Pright = 0;

// variables that will be controlled by the engineer, showing if the system is engaged or not.
int Eup = 0;
int Edown = 0;
int Eleft = 0;
int Eright = 0;

OscP5 oscP5;
NetAddress screen;
NetAddress engineer;

void setup(){
    frameRate(60);
    size(1000,1000);
    background(128,128,128);
    smooth();
    textSize(32);
    fill(0,0,0);
    text("Pilot Controls", 400, 32);

    // setting up network connections: My port
    oscP5 = new OscP5(this, 5555);

    // plug from engineer
oscP5.plug(this, "engineerSystems", "/engineer");

// Griffin's Machine
String screenIP = "127.0.0.1";
screen = new NetAddress(screenIP, 4444);

// Tucker's Machine
String engiIP = "192.168.137.99";
engineer = new NetAddress(engiIP, 4444);

// checking for joystick controlled by arduino
if(joystick){
    joy = new Serial(this, Serial.list()[0], 9600);
}
}

void draw(){
    // controls with the joystick if one is connected
    if(joystick){
        joyControls();
    }

    // Lights up the indicators if the systems have been enabled by the engineer
    drawIndicator();

    // Lights up buttons and controls the signal variables to send to the engineer and screen
    buttonControls();

    // sends which thrusters are active to the screen and engineer
    OscMessage m = new OscMessage("/pilot");
    m.add(Pup);
    m.add(Pdown);
    m.add(Pleft);
    m.add(Pright);
    oscP5.send(m, screen);
    oscP5.send(m, engineer);
}

// Function to draw the if the systems are enabled or not.
void drawIndicator(){
    // up indicator
    if (Eup == 1){
        fill(255,0,0);
    } else {
        fill(64,0,0);
    }
}
```cpp
void buttonControls()
{
    // up
    if (upb != 0)
    {
        if (Eup == 1)
        {
            fill(0, 255, 0);
            Pup = 1;
        }
        else
        {
            fill(255, 255, 0);
            Pup = 0;
        }
    }
    else
    {
        fill(255, 255, 255);
        Pup = 0;
    }
    ellipse(500, 250, 300, 300);

    // left
    if (leftb != 0)
    {
        if (Eleft == 1)
        {
            fill(0, 255, 0);
            Pleft = 1;
        }
        else
        {
            fill(255, 255, 0);
            Pleft = 0;
        }
    }
    else
    {
        fill(255, 255, 255);
        Pleft = 0;
    }
    ellipse(500, 50, 25, 25);

    // right
    if (Eright == 1)
    {
        fill(0, 255, 0);
        Pred = 1;
    }
    else
    {
        fill(255, 255, 0);
        Pred = 0;
    }
    ellipse(750, 300, 25, 25);

    // down
    if (Edown == 1)
    {
        fill(0, 255, 0);
        Pdown = 1;
    }
    else
    {
        fill(255, 255, 0);
        Pdown = 0;
    }
    ellipse(500, 550, 25, 25);
}
```
/send the signal to the controller machine to consume power

and move ship
  } else {
    fill(255, 255, 0);
    Pleft = 0;
  }
} else {
  fill(255, 255, 255);
  Pleft = 0;
}
ellipse(250, 500, 300, 300);

//right
if (rightb != 0){
  if (Eright == 1){
    fill(0, 255, 0);
    Pright = 1;
    //send the signal to the controller machine to consume power
    and move ship
  } else {
    fill(255, 255, 0);
    Pright = 0;
  }
} else {
  fill(255, 255, 255);
  Pright = 0;
}
ellipse(750, 500, 300, 300);

//down
if (downb != 0){
  if (Edown == 1){
    fill(0, 255, 0);
    Pdown = 1;
    //send the signal to the controller machine to consume power
    and move ship
  } else {
    fill(255, 255, 0);
    Pdown = 0;
  }
} else {
  fill(255, 255, 255);
  Pdown = 0;
}
ellipse(500, 750, 300, 300);

void joyControls(){
}
while(joy.available() > 0) {
    char inByte = joy.readChar();
    //println(inByte);
    if((inByte&0x08)==0x08){ //Up
        upb = 1;
    }
    else{
        upb = 0;
    }
    if((inByte&0x04)==0x04){ //Right
        rightb = 1;
    }
    else{
        rightb = 0;
    }
    if((inByte&0x02)==0x02){ //Back
        downb = 1;
    }
    else{
        downb = 0;
    }
    if((inByte&0x01)==0x01){ //Left
        leftb = 1;
    }
    else{
        leftb = 0;
    }
}

void keyPressed(){
    //main button controls
    if (key == CODED){
        if (keyCode == UP){
            upb = 1;
        }
        if (keyCode == LEFT){
            leftb = 1;
        }
        if (keyCode == RIGHT){
            rightb = 1;
        }
        if (keyCode == DOWN){
            downb = 1;
        }
    }
}
// up indicator control
if (key == 'w' && Eup == 0 && wasdEngi) {
    Eup = 1;
} else if (Eup == 1 && (key == 'w' || key == 'W') && wasdEngi) {
    Eup = 0;
}

// left indicator controls
if (key == 'a' && Elef = 0 && wasdEngi) {
    Elef = 1;
} else if (Elef == 1 && (key == 'a' || key == 'A') && wasdEngi) {
    Elef = 0;
}
if (key == 's' && Edown == 0 && wasdEngi) {
    Edown = 1;
} else if (Edown == 1 && (key == 's' || key == 'S') && wasdEngi) { 
    Edown = 0;
}
if (key == 'd' && Eright == 0 && wasdEngi) {
    Eright = 1;
} else if (Eright == 1 && (key == 'd' || key == 'D') && wasdEngi) {
    Eright = 0;
}

void keyReleased() {
    if (key == CODED) {
        if (keyCode == UP) {
            upb = 0;
        } else if (keyCode == LEFT) {
            leftb = 0;
        } else if (keyCode == DOWN) {
            downb = 0;
        } else if (keyCode == RIGHT) {
            rightb = 0;
        }
    }
}

void engineerSystems(int inup, int inright, int inleft, int indown) {
    Eup = inup;
    Elef = inleft;
    Eright = inright;
    Edown = indown;
}
Appendix M: Engineer Interface Code

import java.awt.Color;
import oscP5.*;
import netP5.*;
import processing.sound.*;
import processing.serial.*;

Color up = new Color(255, 255, 255);
Color left = new Color(255, 255, 255);
Color right = new Color(255, 255, 255);
Color down = new Color(255, 255, 255);

Serial button;

boolean upb = false;
boolean downb = false;
boolean leftb = false;
boolean rightb = false;
boolean keyTrack = false;
boolean test = false;
boolean consume = false;

int leftConsume = 0;
int rightConsume = 0;
int upConsume = 0;
int downConsume = 0;

boolean sendUp = false;
boolean sendRight = false;
boolean sendLeft = false;
boolean sendDown = false;

// buttonA enables and disables arduino controls
boolean buttonA = true;
boolean upA = false;
boolean leftA = false;
boolean rightA = false;
boolean downA = false;

int deadReset = 0;

float power = 255.0;
int health = 255;

int upBInt = 0;
int downBInt = 0;
int rightBInt = 0;
int leftBInt = 0;
int mouseControl = 0;
double distance;

OscP5 oscP5;
OscP5 osc;
NetAddress Pilot;
NetAddress Screen;

//SoundFile soundfile;

void setup()
{
    // set background, resource bars, and text.
    size(1000, 1000);
    background(128, 128, 128);
    smooth();

    //osc initialization.
    oscP5 = new OscP5(this, 4444);
    osc = new OscP5(this,12000); //controls the interface between max
    oscP5.plug(this, "setHealth", "/screen");
    oscP5.plug(this, "setInts", "/pilot");
    osc.plug(this,"result","/test"); //used for interfacing with max
    Pilot = new NetAddress("192.168.137.1", 5555);
    Screen = new NetAddress("192.168.137.1", 4444);

    //soundfile = new SoundFile(this, "laserCut.wav");
    frameRate(24);

    if(buttonA){
        button = new Serial(this, Serial.list()[0], 9600);
    }
}

void draw()
{
    background(128, 128, 128);
    fill(0, 0, 255);
    rect(0, 0, 50, 255);
    fill(255, 0, 0);
    rect(50, 0, 50, 255);
    frameRate(30);
    textSize(32);
    //text(power, 60, 30);
    fill(0, 0, 0);
    text("Health", 250, 57);
text("Power", 250, 27);
fill(0, 0, 255);
rect(0, 0, 50, 255);
fill(255, 0, 0);
rect(50, 0, 50, 255);

cvtColor();

buttonControls();
fill(128, 128, 128);
rect(500, 10, 300, 50);
fill(0, 0, 0);

String words = Double.toString(floor((float)distance));
text(words, 505, 40);
text("Distance", 800, 40);
text("to Target", 800, 70);

cvtColor();
resetPower();
powerDisplay();
healthDisplay();

println(mouseControl);

// OSC networking code.
OscMessage systemson = new OscMessage("/engineer");
  systemson.add(upBInt);
  systemson.add(rightBInt);
  systemson.add(leftBInt);
  systemson.add(downBInt);
  oscP5.send(systemson, Pilot);
  
  OscMessage powerSend = new OscMessage("/engine");
  powerSend.add((int)power);
  oscP5.send(powerSend, Screen);
}

void keyPressed()
{
  if (key == CODED)
  {
    if (keyCode == UP)
    {
      upb = true;

    }
if (keyCode == LEFT) {
    leftb = true;
}
if (keyCode == RIGHT) {
    rightb = true;
}
if (keyCode == DOWN) {
    downb = true;
}
}
}

void keyReleased()
{
    if (key == CODED)
    {
        if (keyCode == UP)
        {
            upb = false;
        }
        if (keyCode == LEFT)
        {
            leftb = false;
        }
        if (keyCode == DOWN)
        {
            downb = false;
        }
        if (keyCode == RIGHT)
        {
            rightb = false;
        }
    }
}

public void result(int valueA) {
    mouseControl = valueA;
}

void setHealth(int in, double dist, int dead)
{
    //println(in);
    health = in;
    distance = dist;
    deadReset = dead;
void setInts(int up, int down, int left, int right)
{
    upConsume = up;
    downConsume = down;
    leftConsume = left;
    rightConsume = right;
}

//reset power and redraw the bar
void resetPower()
{
    //key to reset power
    if (key == 'r' || key == 'R')
    {
        power = 255;
        fill(0, 0, 255);
        rect(0, 0, 50, power);
    }
    if (deadReset == 1)
    {
        power = 255;
    }
}

void healthDisplay()
{
    //convert the health int to a displayable percent.
    //limiter on health
    if (health == 100)
    {
        fill(255, 0, 0);
        rect(50, 0, 50, 255-health);
    }
    if (health < 0)
    {
        health = 0;
    }
    else if (health > 255)
    {
        health = 255;
    }

    int healthWhole = 100*health/255;
    int healthFrac = health%255;
    fill(128, 128, 128);
rect(50, 0, 50, 255 - health);
fill(0, 0, 0);
String three = str(healthWhole);
String four = str(healthFrac);
String Printer2 = three + "." + four + "]%";
text(Printer2, 100, 57);

//damage and health reset for debugging
if (key == 'h')
{
    health --;
}

if (key == 'o')
{
    health = 255;
}

if (health <= 0 && deadReset == 1)
{
    power = 255;
}

void buttonControls()
{
    //stopper to make it so you cant have all 4 buttons down at once.
    if (upb && leftb && rightb && downb)
    {
        keyTrack = true;
        upb = false;
        downb = false;
        rightb = false;
        leftb = false;
    } else
    {
        keyTrack = false;
    }

    //serial code
    if (buttonA){
        while(button.available() > 0) {
            char inByte = button.readChar();
            println((int)inByte);
            if((inBytes0x08)==0x08){//Up
                upb = true;
            } else{
                upb = false;
            }
        }
    } else{
        upb = false;
if((inByte&0x04)==0x04){ //Right
    rightb = true;
} else{
    rightb = false;
}
if((inByte&0x02)==0x02){ //Back
    downb = true;
} else{
    downb = false;
}
if((inByte&0x01)==0x01){ //Left
    leftb = true;
} else{
    leftb = false;
}
}
//end serial code
//button inputs.
//up
if (upb == true && keyTrack == false && power > 0) {
    fill(0, 255, 0);
    //System.out.println(power);
    //send message to Pilot computer if up is on.
    /*if (soundfile.isPlaying() == false)
    {
        //soundfile.play();
    }*/
    if (upConsume != 0)
    {
        power = power - .1;
    }
    fill(0, 255, 0);
} else {
    fill(255, 255, 255);
    //sendUp = false;
}
ellipse(500, 250, 300, 300);

//left
if (leftb == true && keyTrack == false && power > 0)
{
    fill(0, 255, 0);
    //System.out.println(power);
    //send message to Pilot computer if left is on.
    if (leftConsume != 0)
    {
        power = power - .05;
    }
    fill(0, 255, 0);
} else
{
    fill(255, 255, 255);
    //sendLeft = false;
}
ellipse(250, 500, 300, 300);

//right
if (rightb == true && keyTrack == false && power > 0)
{
    fill(0, 255, 0);
    //System.out.println(power);
    //send message to Pilot computer if right is powered
    if (rightConsume != 0)
    {
        power = power - .05;
    }
    fill(0, 255, 0);
} else
{
    fill(255, 255, 255);
}
ellipse(750, 500, 300, 300);

//down
if (downb == true && keyTrack == false && power > 0)
{
    fill(0, 255, 0);
    if (downConsume != 0)
    {
        power = power - .05;
    }
    fill(0, 255, 0);
} else
{
    fill(255, 255, 255);
    //sendDown = false;
}
ellipse(500, 750, 300, 300);
void powerDisplay()
{
    // limiters on health and power.
    if (power < 0)
    {
        power = 0;
    }
    else if (power > 255)
    {
        power = 255;
    }

    fill(128, 128, 128);
    rect(0, 0, 50, 255 - power);

    fill(0, 0, 255);
    rect(100, 1, 150, 30);
    fill(0, 0, 0);
    float powerPercentage = (100 * power/255);
    String stringPower = str(powerPercentage);
    stringPower = stringPower + "00000";
    stringPower = stringPower.substring(0, 6);
    String Printer = (stringPower + "%")
    text(Printer, 100, 27);
}

void convertToInt()
{
    // convert bools to ints for OSC
    if (upb == true)
    {
        upBInt = 1;
    }
    else
    {
        upBInt = 0;
    }

    if (downb == true)
    {
        downBInt = 1;
    }
    else
    {
        downBInt = 0;
    }

    if (rightb == true)
    {
        rightBInt = 1;
    }
} else
{
    rightBInt = 0;
}

if (leftb == true)
{
    leftBInt = 1;
} else
{
    leftBInt = 0;
}
}

void checkNum(int first, int second)
{
    while (first > second)
    {
        rightb = true;
    }
    while (first < second)
    {
        leftb = true;
    }
}
Appendix N: Playtest Signup Sheet

Please sign up for at least one time slot per role
You cannot perform both roles in one play session

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Pilot Role</th>
<th>Engineer Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday 10/1</td>
<td>12:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuesday 10/2</td>
<td>12:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3:30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix O: Revised Pilot Control Code

```c
int count=0;
int deadZone = 150;
int pos2 = 1*(512-deadZone)/3;
int pos3 = 2*(512-deadZone)/3;

void setup() {
    Serial.begin(9600);//Setup serial connection
}

void loop() {
    count=0;
    int x = analogRead(0);
    int y = analogRead(5);

    if(y<(512-deadZone)){//UP
        if(y<(pos2)){
            count+=0x0003;
        }
        else if(y<(pos3)){
            count+=0x0002;
        }
        else{
            count+=0x0001;
        }
    }

    if(y>(512+deadZone)){//DOWN
        if(y>(512+deadZone+pos3)){
            count+=0x000C;
        }
        else if(y>(512+deadZone+pos2)){
            count+=0x0008;
        }
        else{
            count+=0x0004;
        }
    }

    if(x>(512+deadZone)){//LEFT
```
if(x>(512+deadZone+pos3)){
    count+=0x00C0;
} else if(x>(512+deadZone+pos2)){
    count+=0x0080;
} else{
    count+=0x0040;
}
}

if(x<(512-deadZone)){//Right
    if(x<(pos2)){
        count+=0x0030;
    } else if(x<(pos3)){
        count+=0x0020;
    } else{
        count+=0x0010;
    }
}

delay(10);
Serial.println(county);