Microsoft Eye-Tracking Game Development

Solution Approach

The MS Eye Tracking Game Project's goal is to develop a video game where the player is able to play the game exclusively with eye-movements. This game must teach the user how to use the basic gestures of eye tracking technology.

Team Crios

Taryn Burns

Kelsey Nash

Mike Berger

Cole KaTrinh

Matthew Trovillo

Course

CptS 421 Software Design Project 1

Instructor

Aaron S. Crandall

Mentor

Jon Campbell

TABLE OF CONTENTS

I. Introduction

The Microsoft eye-tracking game development project's goal is to develop a video game that can be played exclusively with an eye-tracking interface so that normal users or users with ALS can play the game without the use of a mouse or keyboard. The game will be designed to teach people how to use basic eye-tracking gestures, similar to how Solitaire and Minesweeper were designed to teach how to interact with a computer by using the mouse [3]. By the end of the semester next spring, we should have a product that can be published on the windows 10 store so that any user with eye-tracking hardware can download and play the game.

Our goal for this document is to map out the solution we have designed for our video game. We will begin with an overview of the system, then dive into the Architecture of the game and the User Interface. This document is evolving and is subject to change.

II. System Overview

Gods of Calamity's goal is deliver an experience that intuitively teaches the user how to use the basic gestures of eye gaze interaction. To accomplish this goal, the game must be built around these gestures, essentially binding them to actions in the game. To make Gods of Calamity will provide the context for these actions. The game's features can then be described as follows:

The goal of the game is to protect a small village in the woods from natural disasters. The game is played in a top-down two dimensional perspective. Over time the village will expand,increasing the difficulty by effectively increasing it's hitbox.In addition to the village's expansion, the camera will slowly zoom out until it reaches a prescribed "height".

The objective is to protect the village by dispelling calamities. There are 3 potential calamities; lightning storms, meteor showers and fires. Each calamity must be dispelled in a different way. Meteors are dispelled with quick fixations, lightning storms are dispelled by extended dwells while tracking the path of the storm and fires are dispelled with standard dwells. Meteor showers and lightning storms can create fires when they impact with the forest and fires can spread if left unattended.

As the camera zooms out, calamities will shrink. They will not shrink past a target size, which will be observed when the camera reaches its max prescribed "height". At this point, calamities will spawn more frequently.

The player will attempt to survive as long as possible. The game ends when the village's health hits zero. The village's health is reduced when it is struck by calamities. The village's health can be observed by fixating on the village.

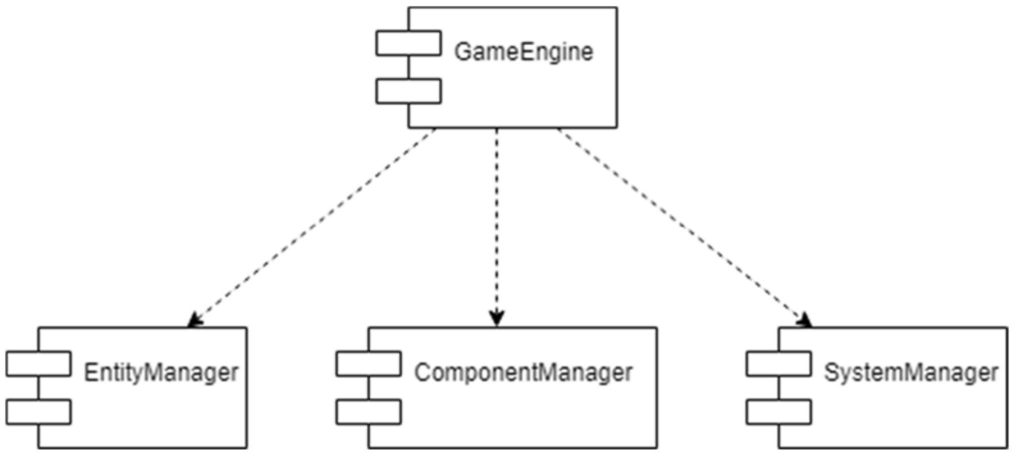
There will be a pause button in the upper right hand corner of the screen and when gazed will display a menu. The menu will allow the player to resume or quit the game.

III. Architecture Design

II.1. Overview

We will be using the Entity-Component System (ECS) pattern for our software architecture. The ECS pattern consists of Entities, Components, and Systems. Entities represent every object in the game, from the village to a weather element. Instead of inheriting attributes from another class, an entity has a list of components which represent attributes like position or health. Systems then operate on the components of the entities, providing game logic. These are the same kinds of elements used in game engines like Unity and Unreal [1].

The Entity-Component System was chosen over other architectures because of its prominence in the game industry and because it avoids problems that can occur with multiple inheritance, by following the Composition over Inheritance principle [2]. Specifically, the pattern fits well for our game because many objects will share similar components that would not make sense to use inheritance with.



The GameEngine will contain our EntityManager, ComponentManager, and SystemManager and provide any setup logic we will need for the game. The EntityManager will contain and manage all of our entities. The ComponentManager will contain and manage all of the many components that will be created. The SystemManager will contain and manage all of the systems that modify the components contained in the ComponentManager. Diagrams breaking down each of the Managers can be found in Appendix A.

II.2. Subsystem Decomposition

- Game Engine
    a) **Description**

  - The Game Engine contains the other managers in the game and also controls all of the game logic and setting up the game.

b) **Concepts and Algorithms Generated**
  - The Game Engine is responsible for initializing other managers and effectively serves as the main program loop for the game. Because it controls all of the game logic, the game manager is responsible for tracking the progress of the game and orchestrating all in-game events. It does this by triggering events that are handled by the Entity Manager, the Component Manager and the System Manager.

c) **Interface Description**
  - The Game Engine sends a command to one of the three Managers that will then perform the dictated task. For example, if the Game Engine determines that a new disaster must be spawned, it commands the EntityManager to create the disaster. When the disaster is created, the Game Engine will command the Component Manager to render it.

Entity Manager

a) **Description**
  - The Entity Manager contains a list of all of the entities in the game. Entities consist of a unique ID and a list of components.

b) **Concepts and Algorithms Generated**
  - We will be using the factory pattern to create the weather elements that threaten the village in the game. Not only do we need different kinds of weather disasters, we also need to create a large number of each. The factory pattern will provide easy weather creation for us.

c) **Interface Description**
  - When a command is received to create a new entity such as a village or disaster, the Entity Manager will facilitate that Entity's construction through it's own subordinate systems and then return the new Entity to the Game Manager.

Component Manager

a) **Description**

- The Component Manager contains a list of all components being used in the game. Each component has an associated entity and is manipulated by a related system.

b) **Concepts and Algorithms Generated**

- The Component Manager handles the abstract representation of objects and systems of the game. The Component Manager binds new entities to their behavior whether that be rendering to the game view or the interactivity of a disaster.

c) **Interface Description**

- When a command is received after the health system updates the health, it will update the village component.

System Manager

a) **Description**

- The System Manager contains a list of all the systems used in the game. These systems manipulate the components associated with entities in the component manager list.

b) **Concepts and Algorithms Generated**

- The System Manager facilitates the execution of individual components. When the System Manager receives the command to execute a specified component's relevant behavior, it will call the proper functions.

c) **Interface Description**

- Systems must be bound and initialized with components. So when the Game Engine calls for the creation of a new object in game all three managers will help to create that object. The job of the System Manager is to define and bind the behavior to the object. The other responsibility of the System Manager is to call the proper behavior on an object when the Game Engine dictates it.

IV. Data design

1) [ScoreBoard Data Structure]

a) Contains an internal list of high scores that is earned at the end of playing a Gods of Calamity that stays after the game ends
b) When game ends, the score is made into a variable that is checked against the elements of the database and sees if there is a score less than it until it reaches a score bigger than it
c) If found, it will take the place of the score that is below the higher one, moving the other scores down until the last score is removed from the list.
d) Plan is to do an XML file that contains the scores and can be read in when game starts in order to check to create the internal score list and can then be checked and rewritten when game is closed when done playing.

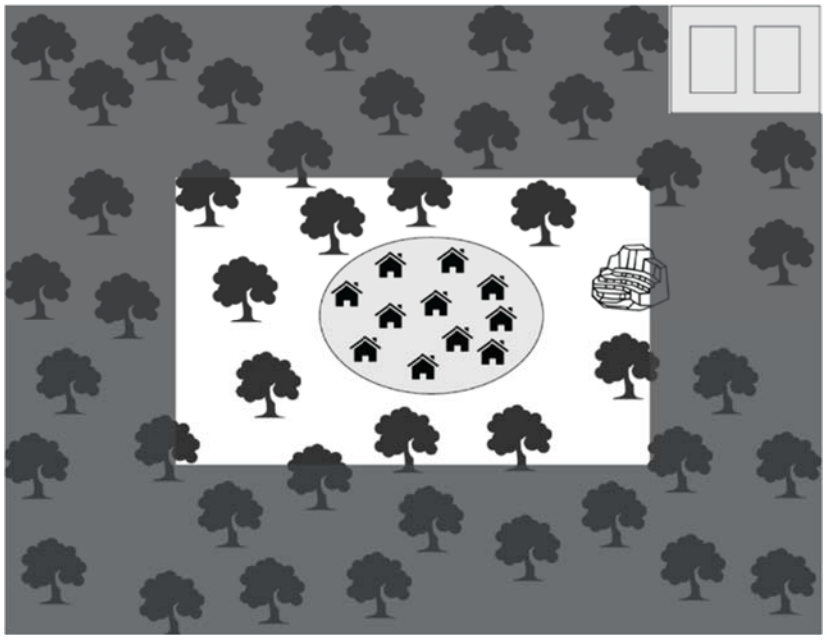
V. User Interface Design

Our user interface will be designed for people first. With limited amount of buttons to make it less complex overall. After anyone downloads the game off of the Microsoft Store, installs it on their computer, then proceeds to launch the game, the game menu will be the first thing to pop up. At this point in time, the entire menu is not designed given that the game design itself is important and weighs heavier in the work load. However, it will be a simple game menu, with a start new game, start save game, resume, and exit button. It will be kept simple and not be crowded so people first can easily access each button.

When downloading the game off of the Microsoft store, everything is included besides the eye tracker hardware. After the game is installed, the player clicks on the game icon in order to launch our game. When the game is launched, the game's title menu will show up on the screen and the game will display by using up the entire screen it's on, like a computer screen.

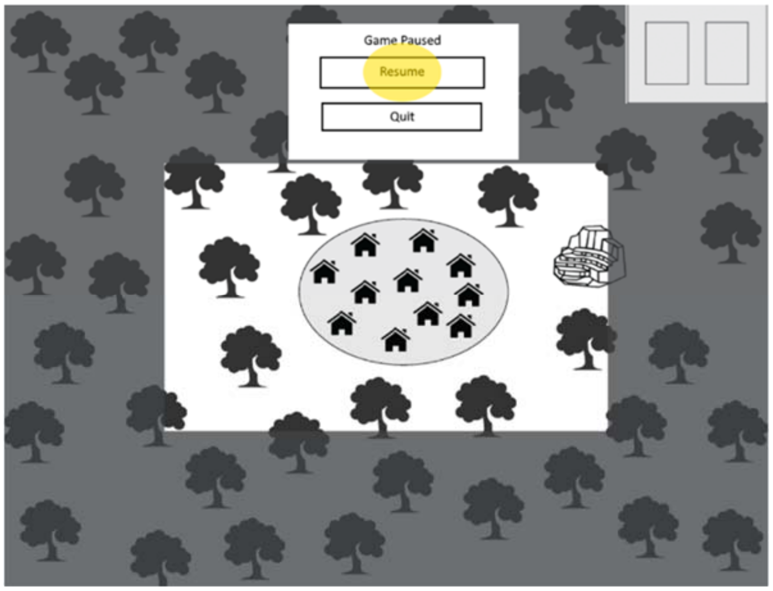
In addition, our gameplays interface will be simple and as least distractive as possible for people first. It is a top-down view of our game's map. We will have a pause button in the top right corner of the screen's display.

In the image to the right is the first portion of the gameplay and simplified:



The map will continuously grow to expand to the game screen's full display as the player continues to play the game and advance through. The game will reveal more land, disasters, and areas the player can use to help expand their village. The map's simple design to there to be user friendly for people with disabilities and to not let their gaze wander too much and to not let the user accidentally interact with something they don't want to. In order to interact with the game more, we will use the different gestures with eye gaze.

Our game will consist of 2D graphics for the alpha prototype.



When the player goes to pause the game and dwells over the pause menu, a screen will pop up and will look similar to the picture shown to the left:

Even with our game pause menu, the interface is still simplified and not super distractive to the player. We are striving to make the interface interact, exciting, yet not super distractive to the players.

Of course it is not shown in the display pictures but our game will consist of colors and slight movements. For example, with the disasters, we will have them slowly disperse as the user interacts with the game by getting smaller and smaller for like the wildfires.

In Appendix A.4, it shows a meteor shower. This disaster will help teach the player the saccade eye movement. If the player is not able to destroy the meteor shower, the land will become unusable. However, if the player does stop the meteor shower, the map will go back to normal.

Each disaster and gaze interaction is shown within our prototype's interface display. In Appendix A.6 and A.7, it shows the lightning storm disaster. This interaction will teach the player the tracking gesture. In order to teach the person this interaction, the lightning storm will slide across the map and the player's gaze will have to follow, or track, the disaster, before it either disperses or destroys a portion of the map.

In Appendix A.9, it shows the game's full map. This reveals when the player has advanced through the game.

Appendix A.10 through A.12 shows what happens if the player does not stop the disasters in time and how they can start fires.

Images of our prototype can be found in the Appendix, starting at Appendix A.4 through A.12.

VI. Summary of the State of This Project

After some consultation and consideration regarding the direction of this project, the Crios team determined that switching development from Unity platform to the Visual Studios Universal Windows Platform (UWP) will be more beneficial to both the upcoming development phases of the game project as well as for the Microsoft team for easier modifications. Team Crios has also settled on designing the game code around C# and the Monogame plugin, which will allow more efficient construction of the game project. The team will also be experimenting with XAML in order to determine whether it will be more beneficial to the project code or not.

Production on the program itself has entered its beginning phases, the environment is currently being set up, which will act as the base for game itself which will interact with/utilize game objects, hazards, etc. Plans for other game elements such as the hazards and game objects have been mocked and will begin production as soon as the environment is set up.

Currently, the initial set-up of the actual UWP XAML is set up on the github workspace and all members have been able to synchronize the project files with their own systems.

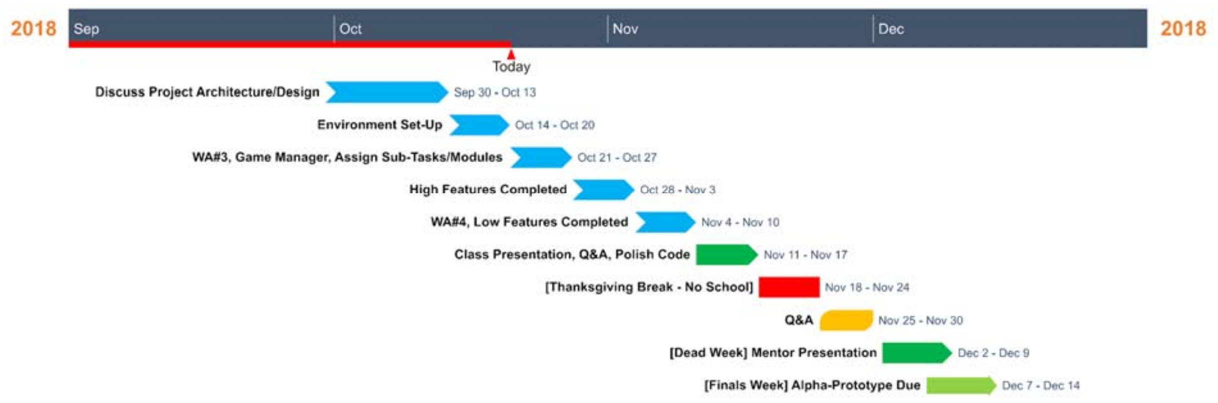
VII. Future Work for This Semester

Moving forward with production, team Crios will be utilizing the basic Gaze interaction library, as well as the Monogame plugin, which will allow for easier and more efficient leverage in terms of utilizing C# to create the video game program.

Team Crios has mocked up a basic overview layout of how to game code will be structured (Referenced in III. Architecture Design), once the basic environment of the game is set up, production on the game engine will begin.

The game engine itself is made of three components (Entity manager, System manager and Component manager), our plan is to evenly split the team labor among each member, based on their strong-suites. Each team member will receive their own branch, which will serve as their individual workspace for the partitioned code they were assigned to program.

Project Timeline:



Member	**Role**
Mike	Game Designer / Programmer
Kelsey	Game Designer / Programmer
Matt	Game Designer / Programmer
Cong	Programmer / Artist
Tayrn	Programmer / Artist

VIII. Glossary

Code reuse - the use of existing code to build new systems.

Component - describes a specific aspect of an entity. Manipulated by systems.

Composition over Inheritance principle - the principle that classes should achieve polymorphic behavior and code reuse by containing instances of other classes that implement the desired functionality rather than inheriting from other classes.

Entity - a game object that consists of a list of components and a unique id.

Game logic - code that controls the game. Could move characters and enemies, update graphics, etc

Instance - a concrete occurance of any object or class.

Polymorphism - a feature of a programming language that allows routines to use variables of different types at different times.

System (ECS) - game logic that operates on related groups of components.

IX. References

(Dutoit, 2010) , 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice Hall, 2010.

[1] K. Hoffman, “Distributed Entity Component System Architecture in the Cloud,” *Medium*, 22-Apr-2018. [Online]. Available: https://medium.com/@KevinHoffman/distributed-entity-component-system-architecture-in-the-cloud-326a94394e84. [Accessed: 18-Oct-2018].

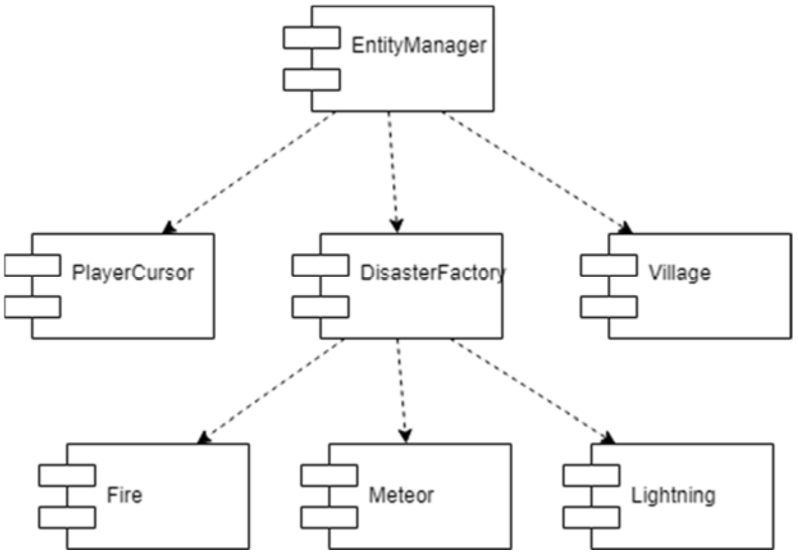
[2] J. Lowrey, “Videogame Architecture,” *James R Lowrey*, 13-Jul-2018. [Online]. Available: https://jarlowrey.com/blog/game-architecture. [Accessed: 18-Oct-2018].

Cite your references here.
For the papers you cite give the authors, the title of the article, the journal name, journal volume number, date of publication and inclusive page numbers. Giving only the URL for the journal is not appropriate. You should use either IEEE or Chicago style formatting for your citations
For the websites, give the title, author (if applicable) and the website URL.

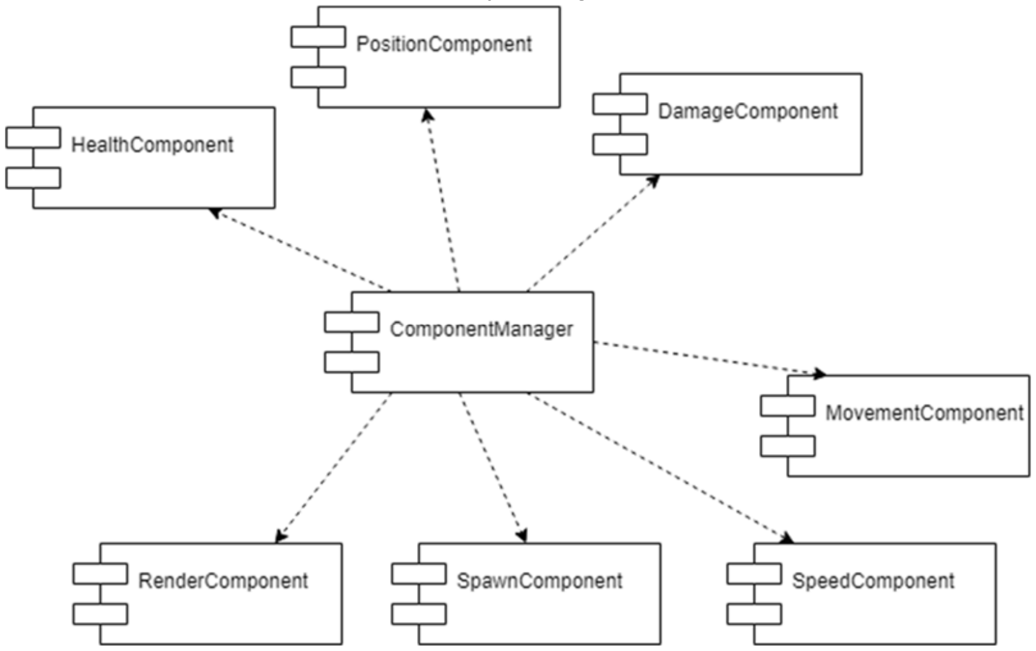
X. Appendices

Any larger images, charts, or external materials should be put into appendices. These are attached at the end of the document so the main materials are kept closer together and the overall flow of the document is preserved. If you include 4 pages of spreadsheets in the middle of a section, it makes it very difficult to track the flow of your presentation. Instead, those sheets go in Appendix [X] and are referred to by the earlier document.
You may have as many appendices as you need for the document to make sense.

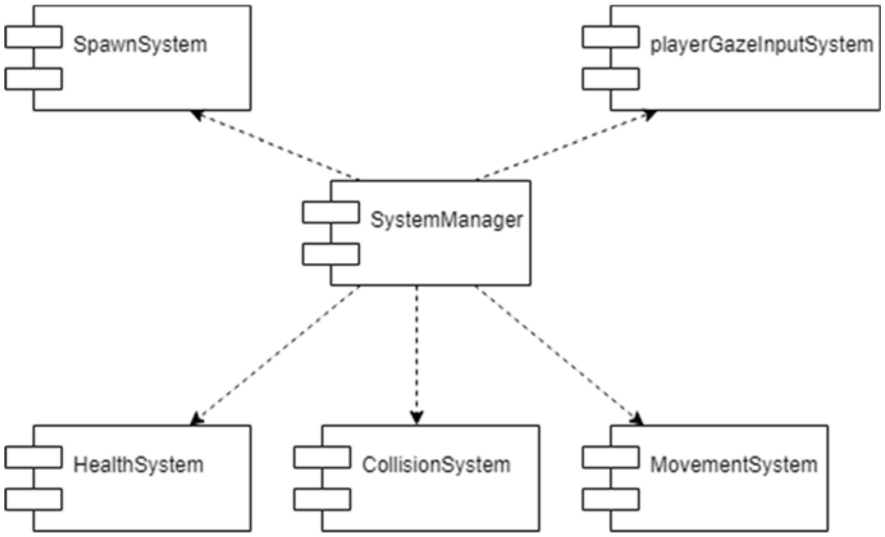
Appendix A: Architecture Component Diagrams



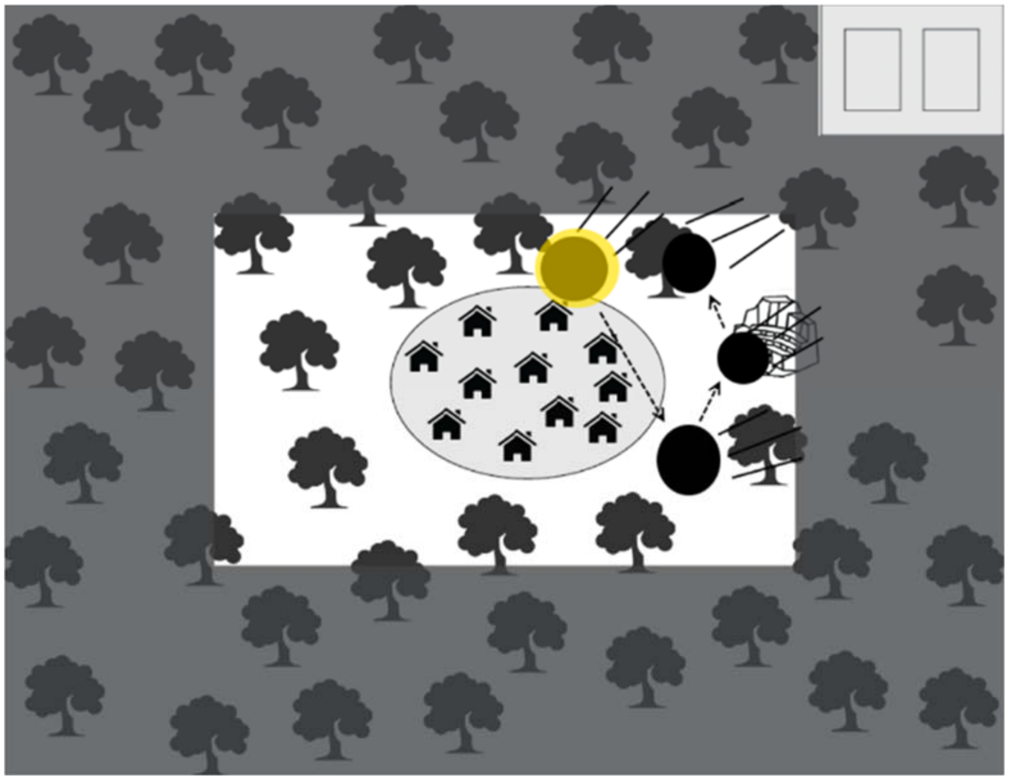
Appendix A1. EntityManager Components



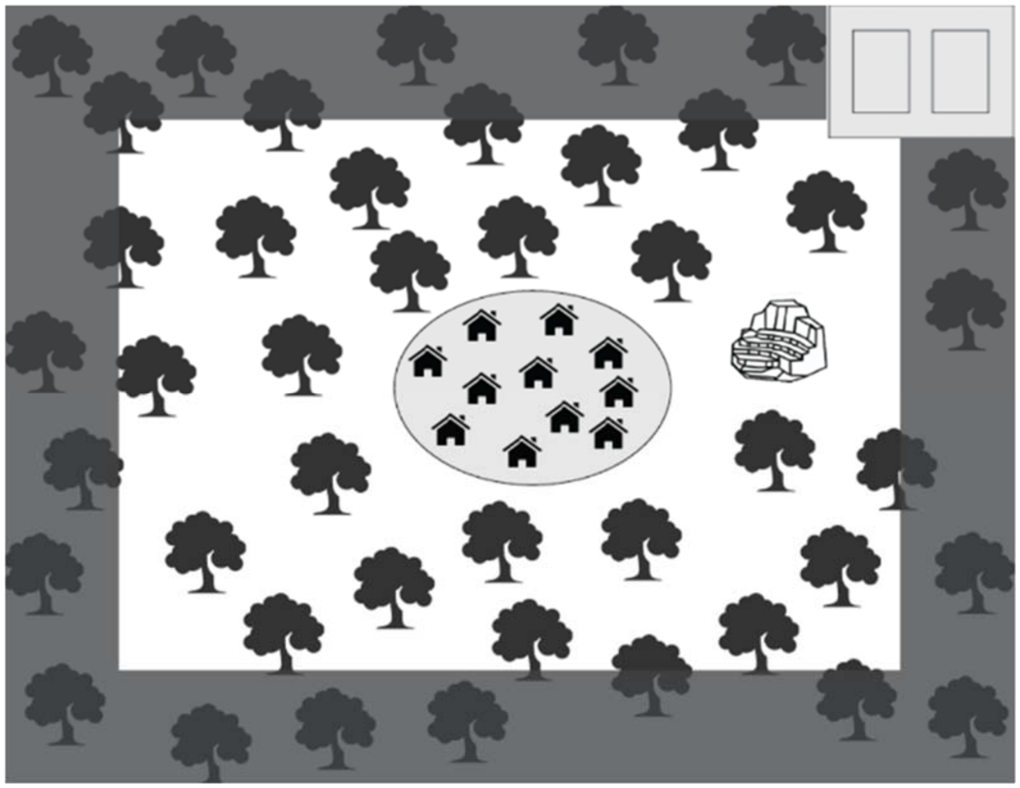
Appendix A2. ComponentManager Components

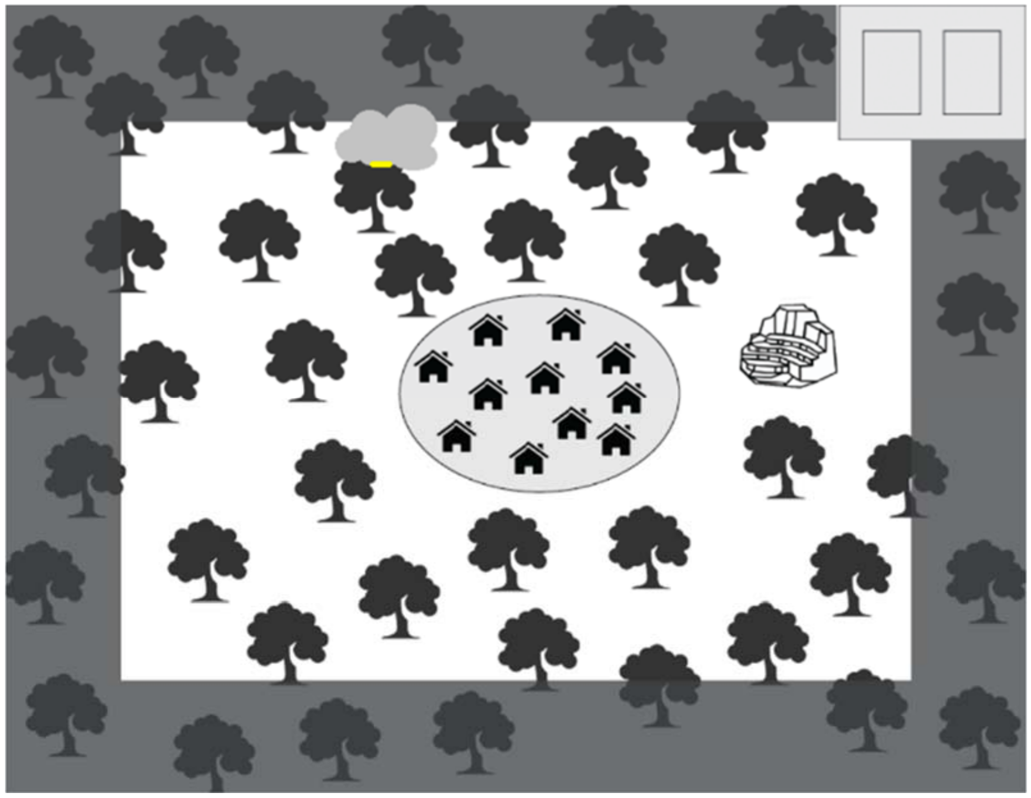


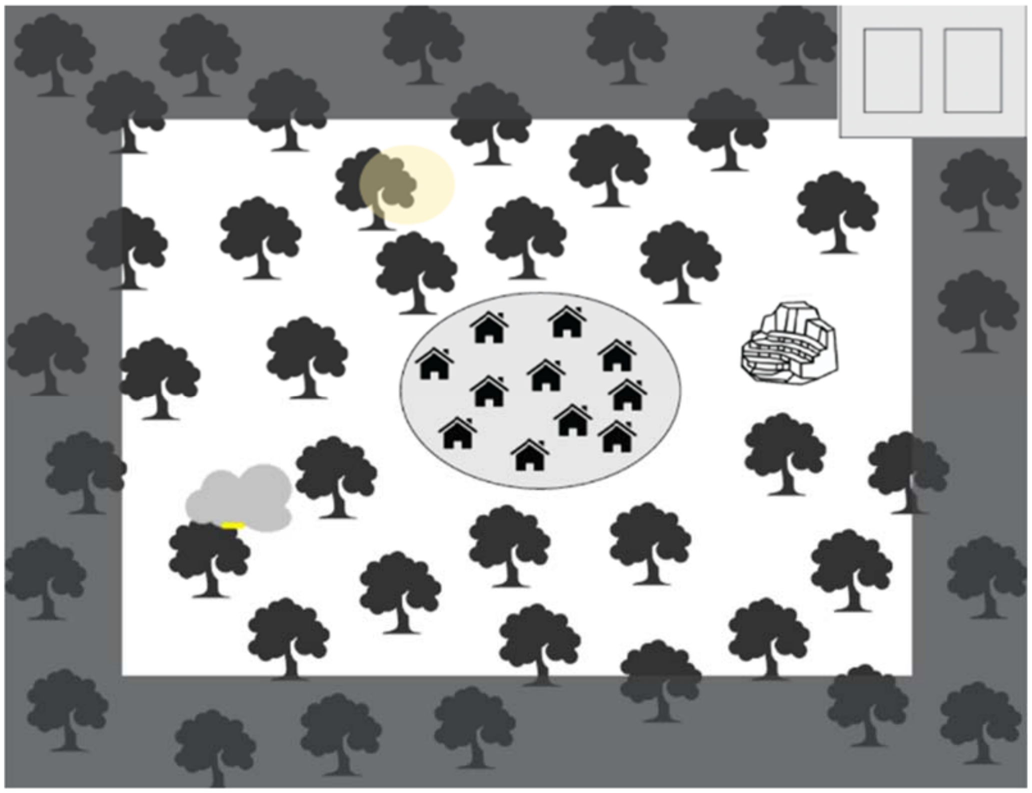
Appendix A3. SystemManager Components

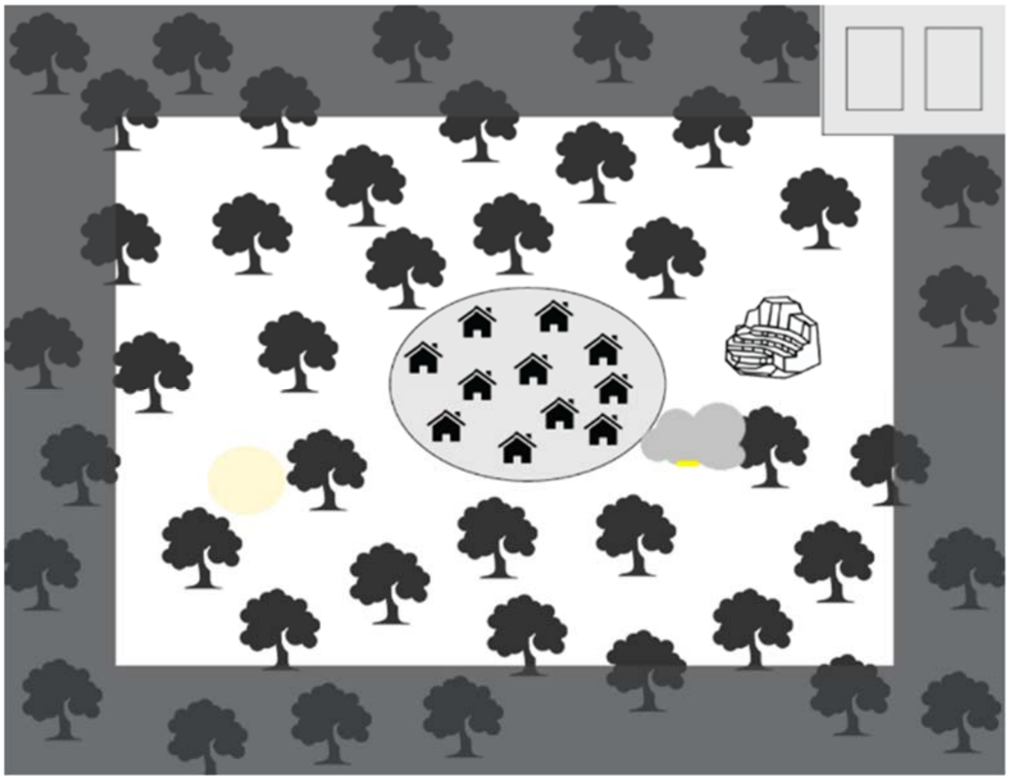


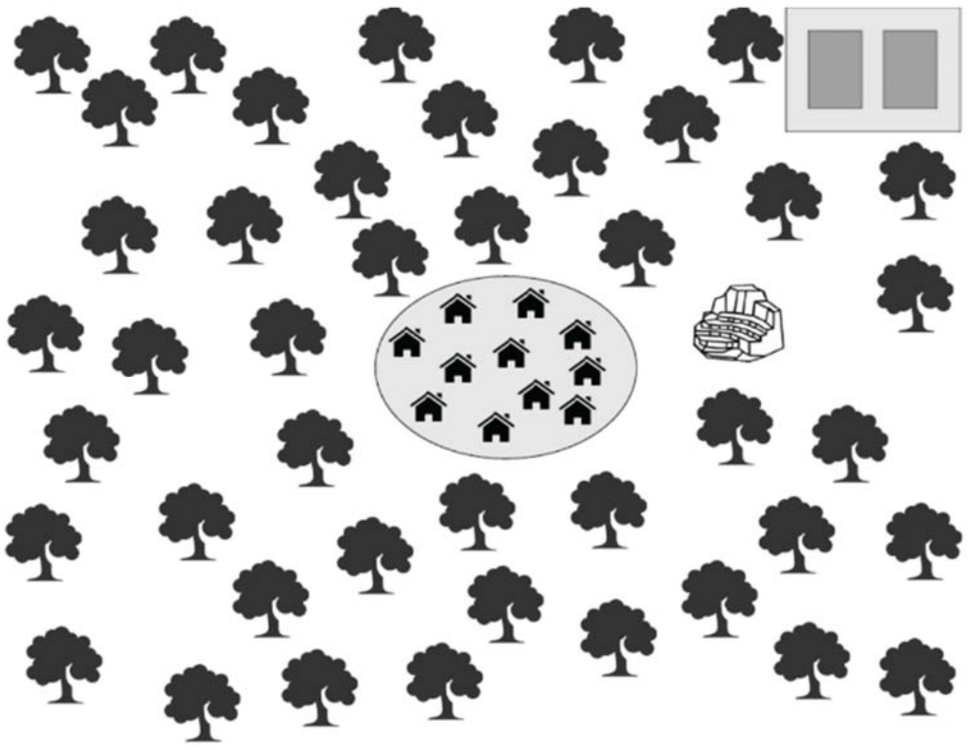
Appendix A4. Meteor Shower Disaster Plus Gaze Interaction


Appendix A.5 The Game Map Expands Over Time and Gameplay

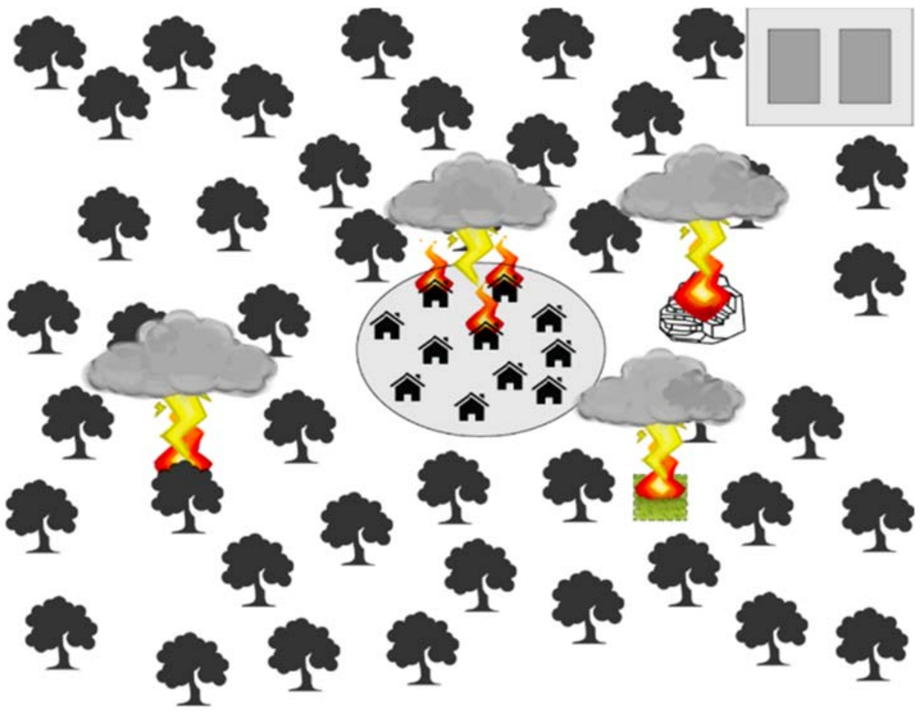

Appendix A.6 Lightning Storm Disaster


Appendix A.7 Lightning Storm Moves Around the Map

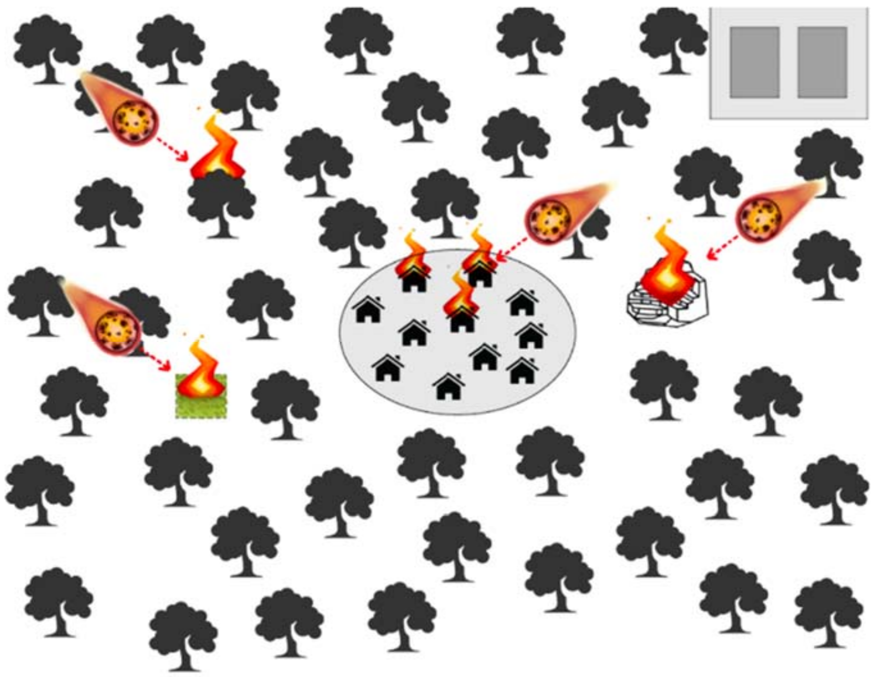

Appendix A.8 Lightning Storm Moves Around Map For A Second Time



Appendix A.9 Full Game Map



Appendix A.10 Lightning Storm Disasters Starting Wildfires



Appendix A.11 Meteor Shower Disaster Starting Wildfires