Red/Blue Video Game for Attack and Defense Analysis Based on the Seeker's Dilemma

Spring 2024

College of Engineering and Computing

Department of Electrical and Computer Engineering

ECE 449: Senior Design

Advised by Dr. Jamieson and Dr. Bhunia

Obed Amaning-Yeboah amanino@miamioh.edu
Computer Engineering | Miami University

Jared Butler butler57@miamioh.edu
Computer Engineering | Miami University

Deniz Misirlioglu misirld@miamioh.edu
Computer Engineering | Miami University

Riley Taylor taylo550@miamioh.edu
Computer Engineering | Miami University

Table of Contents

Table of Figures

Introduction

This project explores the dynamics of a unique Hide&Seek game as a human-based simulation for cybersecurity practices as defined by Chapman *et.al.* [11]. Hide&Seek, within the context of cybersecurity, can be seen as an ongoing game of cat-and-mouse. The red player, the cat, takes the role of the seeker, while the blue player, the mouse, is the hider. This project aims to emulate a game theory model known as “The Seeker’s Dilemma”, a mathematical representation of Hide&Seek on a graph.

In our case, time is a limited resource to the seeker which applies a constraint in our game. Exploration versus exploitation is continuously seen in the nature of this game, as the seeker or hider must decide to exploit the current information known or explore to gain new information and find better alternatives. Is the optimal strategy for the hider to stay in a specific location? To test this, we built a two-player online Hide&Seek game. While playing the game, the positions of both players are constantly being recorded. A heatmap is then generated, visually displaying the location of the hider and seeker while also displaying the amount of time spent at a given location. Analyzing this information can lead to conclusions about the behavior of players in terms of strategy when either taking the role of a seeker or hider.

Organization of the Report

Page 6 - Background: The background is the foundation of the problem and explains the dynamics of "The Seeker's Dilemma."

Page 6 - Game Engine Considerations: Discussing the choices of game engines and features of Unreal Engine 5. The important features are also discussed such as Epic Online Services, Steam, and the use of blueprints.

Page 9 - Gameplay Framework: Gameplay Framework is structured by discussing the major classes used in Unreal Engine. The section ends with our resulting coordinate heatmap from both previous and final versions of the game.

Page 10 - Creation of the Heatmap: A visual representation of our data collected.

Page 12 - Replication: Replication discusses the multiplayer backend and how we implemented it. We also cover remote procedure calls and low-level multiplayer coding.

Page 14 - Final Multiplayer Decision: Deciding on what multiplayer framework to work with.

Page 15 - The Game: The game section covers the early, middle, and late development of our project.

Page 18 - Source Control: Source control explains what service we used to store iterations of the entire project.

Page 19 - Data Collection and Analysis: Methods of data collection and analysis using the Markov Chain model.

Page 20 - Experimental Setup and Results: Observations and discussions on the results of the project. This section covers each graph and the observable results.

Page 27 - Future Work: Future work explains what is next for our project, covering what will be done in the future.

Page 27 - Team Reflection: Team reflection describes the reflection of each team member throughout the entire project.

Page 27 - Ethical Responsibility: The ethical responsibility of engineers to commit to upholding ethical practices in our work.

Page 28 - Conclusion: Concluding the project, summarizing each section of the project.

Background

The background of the project revolves around the exploration of the dynamics of a Hide&Seek game, serving as a simulation for cybersecurity and hardware trojan practices. The rules are based on the principles of “The Seeker’s Dilemma.” “The Seeker’s Dilemma” is Hide&Seek on a graph that is based on game theory. The principles of the seeker’s dilemma are, that the hardware trojans are hidden from the detection algorithm, the number of objects that are hidden is unknown, and the game is limited to two players. Knowing these principles, the rules of our Hide&Seek game align with the principles. The design and implementation of our game is unique, but are roughly based on The Nun’s on the Run, Dracula, and Ultimate Chicken Horse. This project aims to analyze player behavior in both seeker and hider roles by continuously recording their positions and generating a heatmap. This analysis serves as a basis for concluding effective strategies in the context of the Seeker's Dilemma.

Game Engine Considerations

Overview of Unity or Unreal Engine

To create a modern multiplayer game, the use of a game engine is heavily needed as it provides features that would take too much time and effort to implement from scratch. We narrowed the engine choice down to two options, Unreal Engine and Unity. Both choices required significant coding skills in C# or C++. Unreal Engine is the more friendly option as it is based in C++ and our group has experience in the language. This engine also has a feature called blueprints, which allows developers to use all of Unreal Engine’s utilities via an easy-to-learn visual scripting system.

We use Unreal Engine because of Epic’s integrated online services called EOS(Epic Online Services). There are many useful and powerful dev tools available, which makes setting up the framework for multiplayer easy. An example of a tool that is used is the multiplayer session setup which uses peer-to-peer networking between players.

It was difficult to find many positives to Unity compared to Unreal Engine. The programming language, community and company support, and easier-to-understand and use interfaces were the main reasons for choosing Unreal Engine over Unity.

Unreal Engine Age Discussion

When discussing potential engine age issues, we concluded that Unreal Engine 4.0 could be outdated, and support for its features may no longer be useful for our purpose. We did consider Unreal 4.0, but figured it would be best to develop using one of the more recent builds of Unreal,

this decision did cost us some time when developing different versions of the game The newest version, Unreal Engine 5.3, was released 2 weeks to the start of the project. This led to less content and documentation provided for its use. If we decided to use the most recent version of 5.3, we would have to implement every feature without the help of plugins and community support. Since the project is time-sensitive, it is best not to work with this version. In conclusion, we decided to pursue Unreal Engine 5.2.1 as it is between 4.0 and 5.3, and there were useful, developing plugins available for older versions of Unreal Engine 5.

After some time developing, we concluded that even Unreal Engine 5.2.1 was too new, as a lot of features were required to be made without reference to previously made material readily available and brought forth by the Unreal Engine community. Most of the current features the game includes were made based on ideas in version 4.0 of Unreal Engine. While implementing, we struggled to find some specific settings that only Unreal Engine 4.0 provided which resulted in significant workarounds and new ideas entirely.

Epic Online Services (EOS) and Steam Integration

With the game engine in mind, the decision to use a client-server model was chosen, as opposed to having a dedicated server which was deemed unnecessary. In the client model, one player acts as the server and allows a player to connect, deeming them the client. We had the choice of either EOS or Steam's integration services. Our first iteration of server-client implementation utilized EOS. The reason for choosing EOS initially was its easy integration into Unreal Engine. Since EOS was created by Epic Games, using this service was smooth and had no compatibility issues making it straightforward to implement. When a player hosts a game, a new server is opened and all servers can be monitored in the developer portal, allowing for full control of the system.

Scalability was a major goal of ours for the game. Making the game scalable meant that we could add aspects without the worry of technical debt. We realized that EOS had scalability issues, especially when attempting to create a lobby system, and for that reason we switched our multiplayer framework to Steam. Steam proved to be the more user-friendly multiplayer system as a result.

Blueprints

The blueprint visual scripting system in Unreal Engine utilizes nodes that are friendly and easier to use than C++. Since our team was completely new to Unreal Engine, we used blueprints for all game modes, player states, game states, playable characters, and map geometry. Blueprints allow for agile development, as changes can be made rapidly and results can be easily tested in the play-in editor window. The figure below shows an example of a blueprint that was made to increase a player's movement speed which is being tracked in the game state.

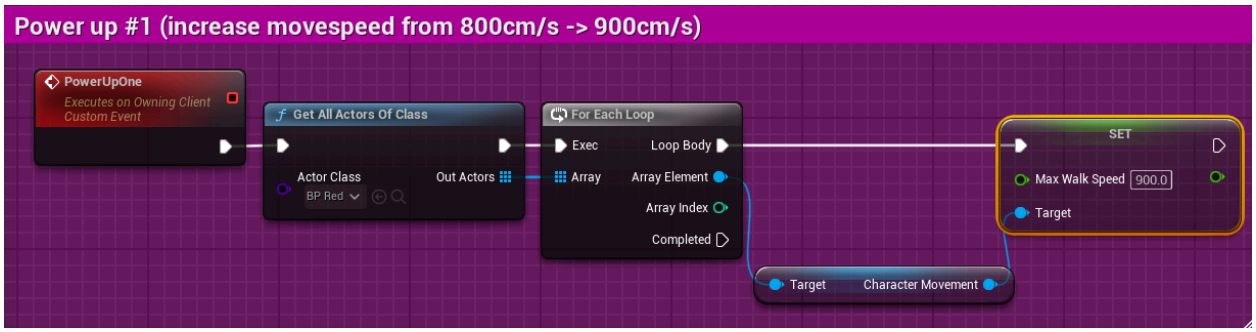

Figure 1: Blueprint visual script for a single power-up.

Figure 1 shows 5 different nodes, each connecting one another in order from left to right. The white lines signify the program execution order, the output of one node will propagate into the input of another node. The final node has no output meaning that execution will terminate as it is the end of the function.

The first node is a custom event, similar to a function in other programming languages. This event allows for a custom name and parameters. It also has properties that allow the event to be replicated on the client if called by the server. The second node is an access node that retrieves the actor that requires property changes. The third node is a standard for each loop that takes in all actors in an array from the previous node. Since there is only ever a single actor with that specified class, the array will be of size 1 always, but for each loop is a failsafe method of getting all array elements if needed. The fourth node is a variable node, it accesses the variable associated with character movement and will set that movement in the final set node with new parameters.

Gameplay Framework

The gameplay framework in Unreal Engine is similar to classes in C++. Aspects of these classes have different dynamics of shared information and communication between server and client. It is important to know how each class is supposed to work in a multiplayer setting because replication is a major issue as the gameplay scales. Adding new features adds additional complexity and these classes allow our game to be more modular.

The Game Mode

In Unreal Engine, a game mode simply defines the game's set of rules. These rules include how players join the game, select their characters, and the logic used throughout the game. This class is only replicated on the server, this is important because it does not allow the client to change the rules of the game unexpectedly. We use this class to define how the game runs and sets timers stored in the game state. When certain conditions are met, it will execute owning client replication events such as disabling and enabling movement and power-ups. The game and associated variables are also able to be restarted in this class.

The Game State

The game state keeps track of the state of the current game mode and current players connected. This class is replicated for all clients, which means all users can access functions and variables in the game state. Our game state controls the spawn points for each player, which player chooses which role, and how their movement is defined given the game’s rules. Any game time-related variables are defined here, as they need to be replicated to clients so information can be updated appropriately.

The Player State

The player state is useful in obtaining information about specific players and their roles in the game. This class is replicated for both the server and clients. We utilize this class to produce player coordinate arrays that are used in producing our hider and seeker heatmaps.

Creation of the Heatmap

The player's coordinates are recorded in hider and seeker arrays stored in the player state at a rate of ten positions per second. For our case, we do not record the z-axis as the map does not have much verticality. A concern we had was the rate at which the coordinate data is appended, but the file output is a comma-separated file, so the final file size of each exported CSV file is not large enough to require adjustments to the data rate. Currently, the user presses 'K' to save heatmap coordinates, and the CSV file will be appended to the user's working or game directory under HAS_Positions. Since only coordinates are exported, a visual representation of the heatmap is created using Python's graphing capabilities.

The Python script is designed for the analysis of player movement within a game environment. It processes the spatial data representing player locations and generates a heatmap to visualize the frequency and duration of player presence in various areas of a game map. The area of the heatmap is limited to the actual area of the playable map. Areas with higher player activity are easily identifiable, which will be valuable for:

- Identifying strategic locations within the game.
- Balancing the game by modifying areas that are either overused or underused.
- Enhancing player experience by understanding player behavior and preferences.

Figure 2 shows a previous example heatmap of the seeker player during a short gameplay test. This heatmap is an older version of the final heatmap that we used for our results. It used a 'YlGnBu' color map, which stands for yellow, green, and blue. These color intensities are used to represent different magnitudes of values in the heatmap. Here's what each color represents in this context:

Yellow: This is the lower end of the scale. Yellow areas on the heatmap represent lower values or lower density of data points. In the heatmap, these are areas with fewer occurrences or lower coordinate values.

Green: As you move up the scale, the color shifts from yellow to various shades of green. Green areas denote medium values or a medium density of data points. This is a middle range between the highest and lowest values.

Blue: Blue is at the higher end of the scale. Dark blue areas represent high values or a high density of data points. In the heatmap, these are the areas with the most occurrences of coordinate values.

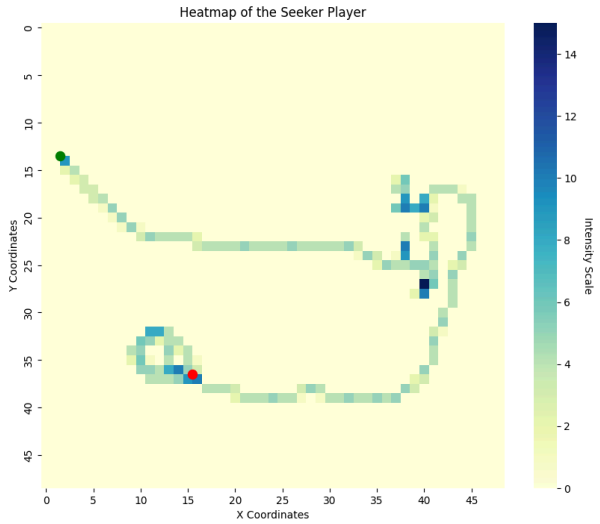

Figure 2a: Previous Heatmap of the seeker player in previous map.

With accessibility in mind, we decided to enhance our heatmap to be three-dimensional, allowing the intensity scale to be visualized by colors and elevation. To quickly identify areas of interest, we made a more visually appealing heatmap with appropriate color-coded node numbers attached. Below shows the new and improved three-dimensional heatmap for our specific map named Valleys of all seeker player position data. The heatmap does not display the exact paths taken by each player, but rather the amount of time spent at certain locations based on the number of recorded positions every second.

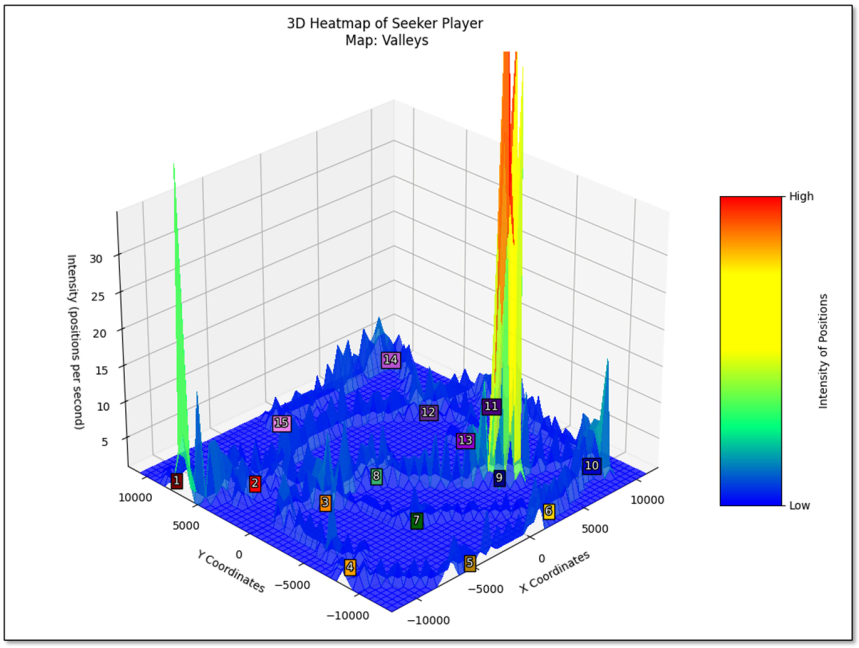

Figure 2b: Final 3D Heatmap of the seeker player on Valleys.

Replication

Replication is how the client and server interact with each other and it deals with every aspect of the game including playable characters, variables, and nodes in each class, and delay in every action. Replication was one of the most difficult issues we had to overcome and it was important to understand when creating a multiplayer game. Several strategies were used to make the game responsive and reduce replication issues.

Remote Procedure Calls (RPC)

RPC, or Remote Procedure Calls, were used throughout the implementation of this project. Unreal Engine utilizes RPC, sending events from the client to the server, server to the client, or a specific group. Unreal Engine allows RPC to work under 3 rules, running said task on server, owning client, or multicast. In this project, replication was needed in almost every implementation scenario, ranging from the UI seen on the screen to the movement of another player. Many instances were encountered where an implementation was seen only on the server side, but not seen on the client side. Because of this, RPC was necessary to implement a majority of the features. If a function requires replication, but cannot inherently replicate, it will first call a custom event that will replicate on the server, which then calls another custom event replicating multicast, or on all clients. This workaround allowed for the game variables and features to be shared across the server and the client such as the spawning of players and the countdown timer for the game to start.

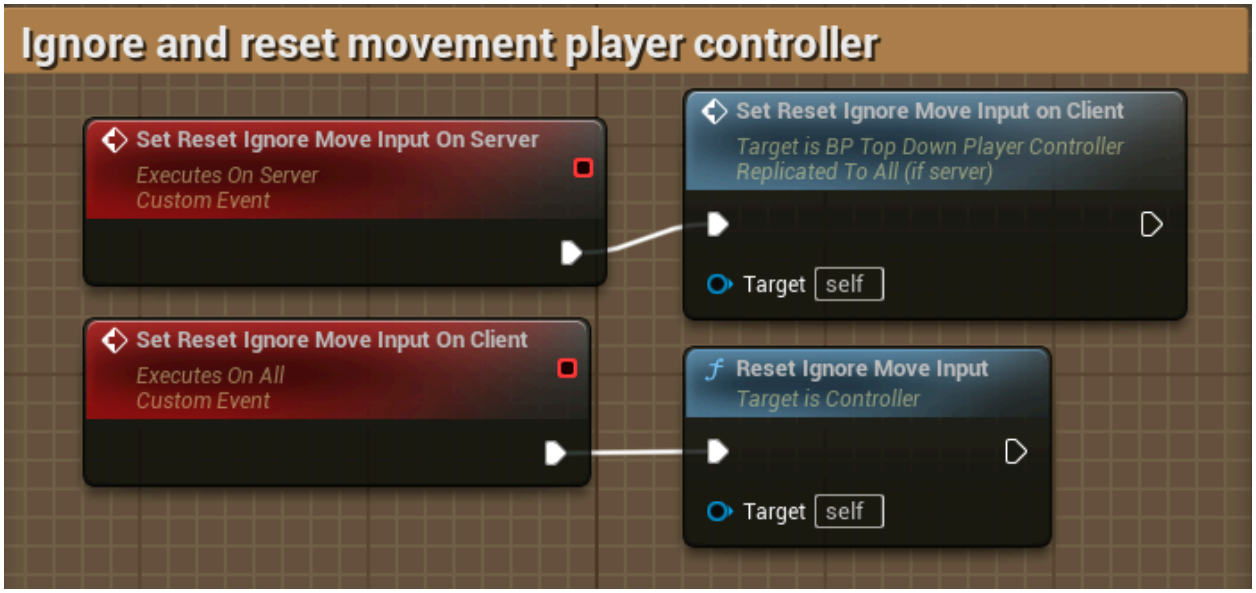

Figure 3: Blueprint representation of RPC.

First Multiplayer Approach

We first tried to implement multiplayer at a high level, within the blueprints themselves using the Unreal framework. This became problematic as our version of Unreal Engine was too new and required new methods of implementation. We then attempted a lower-level approach for dealing with multiplayer sessions. This process included full multiplayer support in C++ using the session create and destroy functionality. When a session is created, it is also replicated onto the developer portal where we have access to details about the server and its status. This resolved our issues with the client not being able to interact with the server's session. After a session is vacated, it is destroyed promptly so that the server list does not build up in number. This is important because for users to connect, there must be servers that are not vacant and are set to be destroyed. To interact with the low-level code, we used widgets that utilize buttons to create and join sessions on our main menu. These buttons work similarly to function calls, when one is pressed it will call the low-level code and run the appropriate function associated with it. The two images below show the early development of multiplayer sessions and a successful connection as server and client as well as the connection being shown in the developer portal with two players connected.

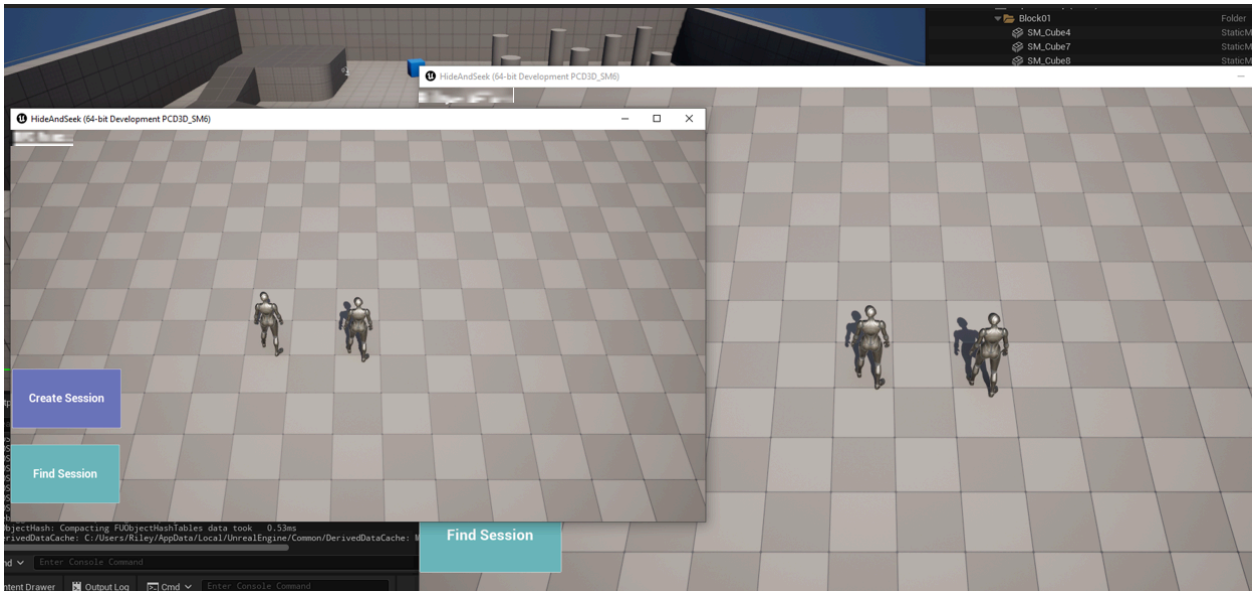

Figure 4: Successful connection with the server and client model with EOS.

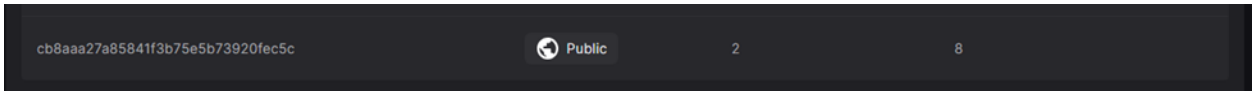

Figure 5a: EOS developer portal server with connections shown.

Final Multiplayer Decision

The Unreal framework was then dropped as we wanted a more scalable approach to our multiplayer system. In its place, we implemented a high-level Steam framework and integrated it into our current system. This improved the shipping of the game as we could ship any version of the game and it would run in Steam without issue because Steam runs the game on their backend which made using the service better than EOS. Instructions to run multiplayer on new machines are now more streamlined and require less prerequisite software. Users are now able to create and join unique servers of their choosing. The lobby system we used also required Steam and would not function properly with the older EOS integration. Figure 5b shows the player searching for a server that has been created by another player.

Figure 5b: Steam server browser from the main menu.

The Game

Early Development Stage

We needed an initial plan for a game, so we brainstormed different ideas and weighed them with viability, originality, and fun. The game was initially a third-person top-down but was transitioned into first-person. The first person made strategy much more interesting as it more closely mimics a real-world scenario. There are two distinct roles, the hider and the seeker. The condition to win for the seeker is to catch and interact with the hider. The hider must avoid the seeker for as long as possible. With these parameters, we went forward with creating the game in Unreal Engine and first implemented the core features required for the game. These features included multiplayer functionality, the creation of a main menu screen, and user interfaces that allow the player to login and connect to sessions. Figure 6a shows one of the first iterations of our main menu screen, which was replaced by Figure 6b in later stages of development.

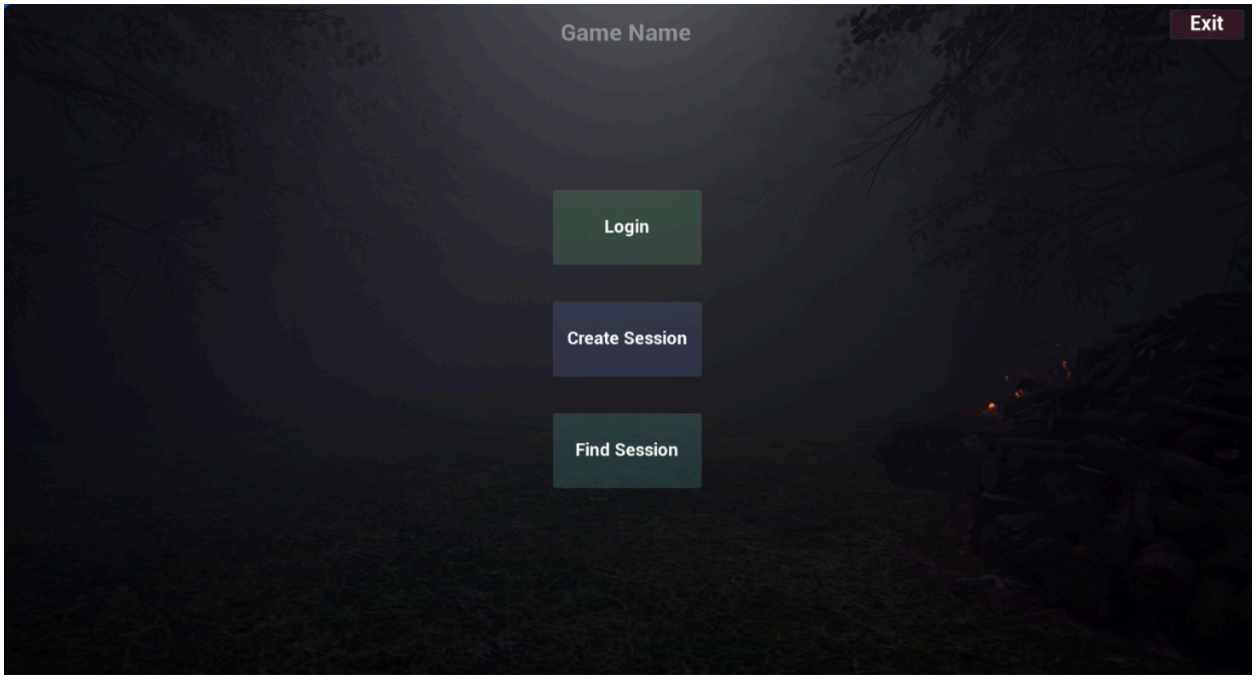

Figure 6a: Previous main menu screen.

To make the game more scalable, we added a player lobby where players may choose their role and map before entering the game. Our previous implementation made it so players were not able to choose the map, and roles were first come first serve. The lobby allows new maps to be added with ease and players are able to formulate strategy before beginning the game. The main menu has been updated to the final version in Figure 7b which now includes Steam integration components and gameplay instructions.

Figure 6b: Final main menu screen.

Middle Development Stage

After completing the initial requirements to make the game functional in multiplayer, we began to implement unique elements to make it more interesting and fun. Designing the game mode was our next important step. Firstly, we had to set the rules of the game and how the game is played. Second, we set the conditions for winning the game for each player. The seeker can interact with the hider, and if done successfully, then the game will conclude and the seeker will win. The hider has a sphere proximity overlap event, when the seeker collides with the sphere then the seeker may interact with the hider. The hider player's win condition is to avoid the seeker player until the timer for seeking has run out. Figure 7a shows a previous version of our role selection screen once the game begins. This has been converted to our current lobby system where players may choose their roles before entering the map. The information bubble provided in previous versions has been moved to the main menu screen. Once the main menu and lobby was completed, we implemented sound design for the ambiance and menu buttons. Figure 7b shows the final version of our lobby including role selection, map selection, and the ability to ready up before beginning the game.

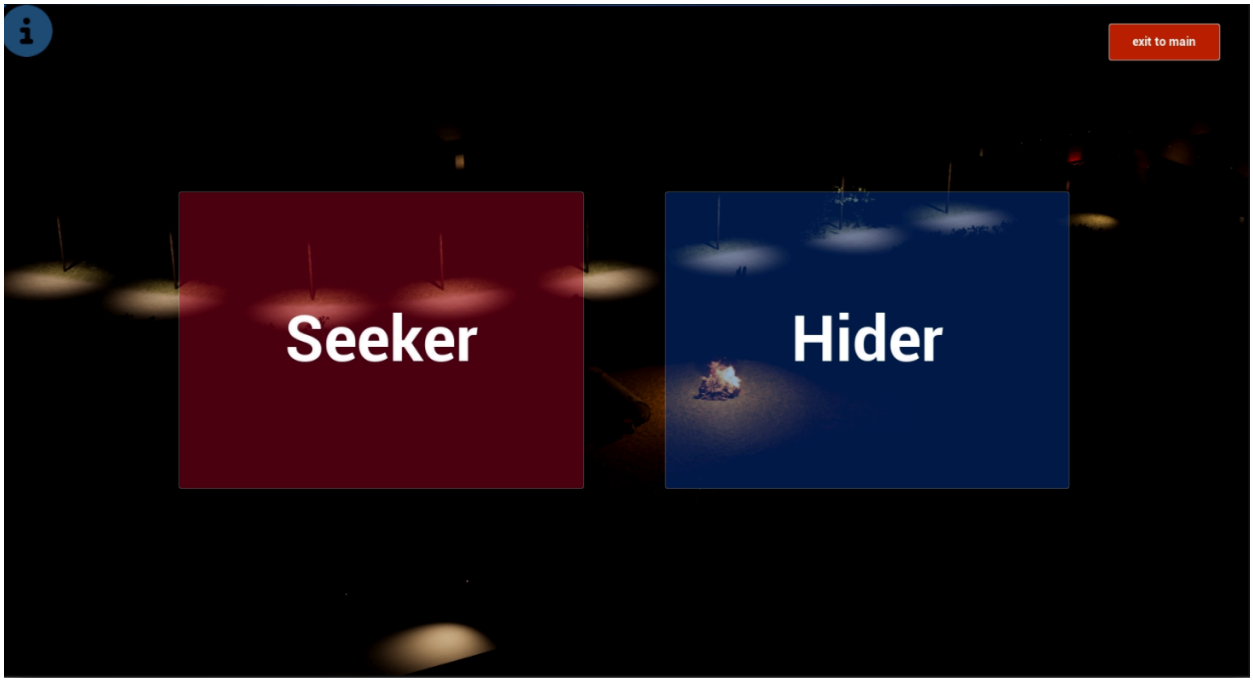

Figure 7a: Previous role selection screen after creating a session.

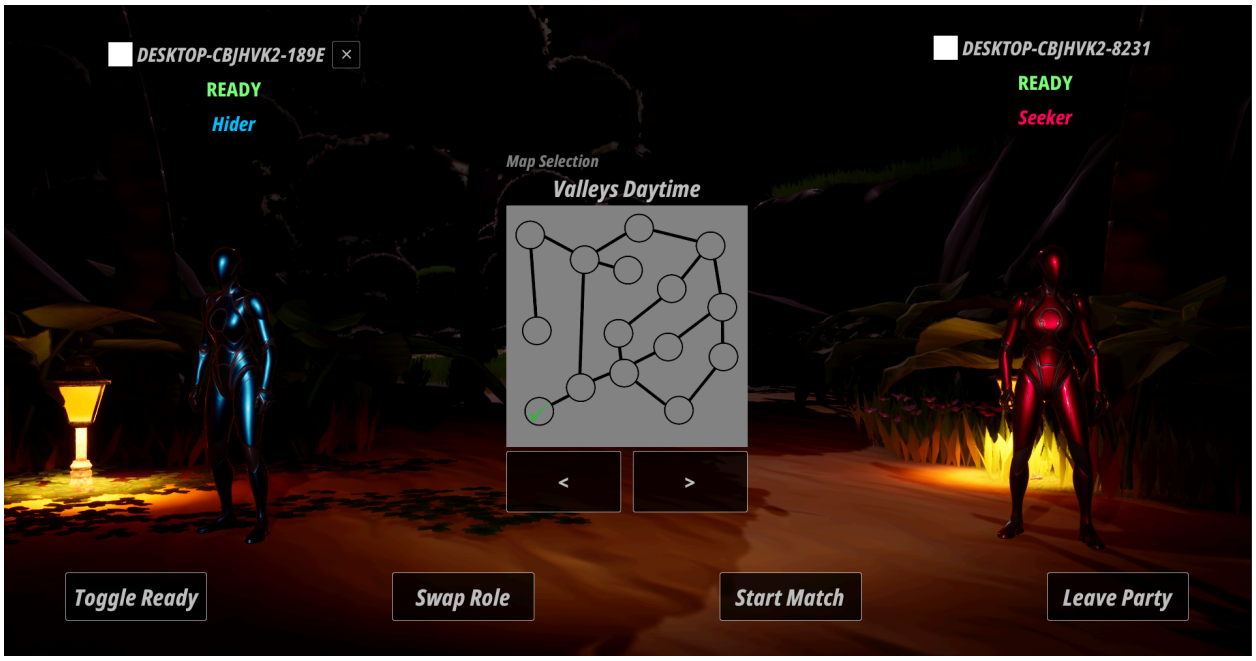

Figure 7b: Final role selection lobby after creating a session.

Late Development Stage

Once we finalized the design of the game, we created three graphs that transitioned into virtual game environments. In addition to using free and open-source assets in our game, we also implemented a procedurally generated foliage system that allowed for automatic resimulation as we shaped each map. The Valleys map includes assets such as hideable bushes and a mansion to make the environment more interactive. These assets add more interesting gameplay elements to the map and more variability to the data that is collected.

We wanted three different graphs to transition into playable maps. To make the maps interesting, we included the map from our first iteration of the game and turned it into a graph. The following two maps we made were Garland Hall at Miami University which included the first and second floors, and a GPIO circuit. Initially, the graphs were designed before creating them as playable maps in the game. Each node and edge is roughly to scale to ensure player positions are recorded correctly. This made for seamless transitions between graph creation and map design.

Source Control

When designing a large-scale project, source control such as GitHub allows for a safe and collaborative environment. GitHub was used continuously to implement changes and add new features to the project, with a branch/merge development strategy. When implementing a new feature to the project, making a branch off the main branch was a safe and secure way to implement a new feature without the chance of breaking the entire project. When a branch was deemed complete, testing was performed making sure other features were still fully functional, and once deemed ready the branch was merged. One main reason for choosing GitHub was the implementation of Github LFS(Large File System), which allows the upload of large files to repositories. Since developing a game requires files that may take up gigabytes of storage, we needed large storage capabilities to have each team member able to work and keep up to date with the project.

Data Collection and Analysis

For this project, an API was created to transfer CSV data from post game to a database in MongoDB. MongoDB was used because it allows for up to 512Mb of free storage. Python was the scripting language of choice to insert and extract all of the data from the database as it allowed for ease of access and serverless API requests. A MongoClient object was created using the specific username and password for the table, and data was inserted. Due to the use of primary collections in MongoDB, a collection was created named HAS-Data, standing for HideandSeek-Data. Inside of the collection, 3 clusters were created which represented the three maps that users are able to play on. The three clusters that represent the data are named HAS-Valley, HAS-Garland, and HAS-GPIO which each have the individual CSV data of every game the testers have played on. When making the entries for the database, we wanted to display all relevant data inside of the database to have straightforward data compilation when analyzing the data. Because of this, for each set of data, 9 attributes were placed in the database. The date of the entry, the score of the hider and seeker represented by a 1 or 0 depending on which one was victorious, and the XYZ positions of the hider and seeker. Because of this data collection was easily completed and analyzing the data as a whole became straightforward.

To retrieve the data, a Python script was written collecting the data from a specific cluster and placing it into an array with each object representing a game with its data. Data from each game's coordinates were then able to be computed and made into a heatmap representing the data as a whole. The collection of data was then used to generate all of the heatmaps for the various maps and incorporate the Markov Chain model analysis.

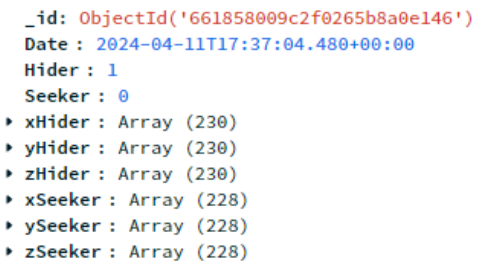

Figure 8: Database entry example.

We used a Markov Chain model to analyze the positional data from the players. The Markov Chain is a useful model because our data follows a stochastic process and each player position is pseudo-random. It also allows us to create a transition matrix that explains player tendencies in moving from one node to another. The transition matrix is then used to create a steady-state distribution that characterizes the long-term behavior of the pool of players in our maps. This distribution mathematically illustrates the amount of time each player spends in each node and the resulting values are normalized.

Experimental Setup

In this project, 16 individuals served as alpha testers to obtain data and test the game. The data collected was then used for final analysis with Python. In each experiment, 2 separate people who have never played the game before were used to test our game and give their opinions. Two laptops were placed on the end of a shared table with the game loaded on both of them. One player was asked to be the host of the game, while the client joined the lobby. Both players agreed on a map and role and began playing the game. Throughout the playthrough of the game, the team was able to capture reactions and first opinions of our game. Also, the game was played on a best of 3 system, making each map have at most 3 rounds while most testers played all 3 maps. The alpha testers have never seen or played the game before, allowing us to collect and analyze unbiased data.

Results

After data collection over an appropriate-sized player population size, we are able to see the interesting behavior of the hider and seeker. The results are of interest to us as player behavior will adjust with knowing the tendencies of other players. Each player is able to freely traverse the virtual environment during the duration of gameplay. The edges of each graph are not finite, necessitating traversal for progression to subsequent nodes. As a result, the analysis presents challenges, as players have the capability to reroute themselves while on an edge and potentially reverse their direction to a previous node. Despite these limitations, we are able to discern patterns and strategies that emerge, revealing the dynamic complexity of the game's graphical representation.

Graph of Valleys

The Valleys graph does not directly represent real-world hardware or application. We utilized this idea to create our own planar graph that included both cycles and end points. The graph in Figure 9a shows that the map has 15 distinct nodes. Each node has their own distinct features to make them unique from one another. Node 9 has a mansion with two floors while node 15 has a single hideable bush, for example. The features inside this graph will have an impact on the player position data as it may take longer for a player to traverse the node.

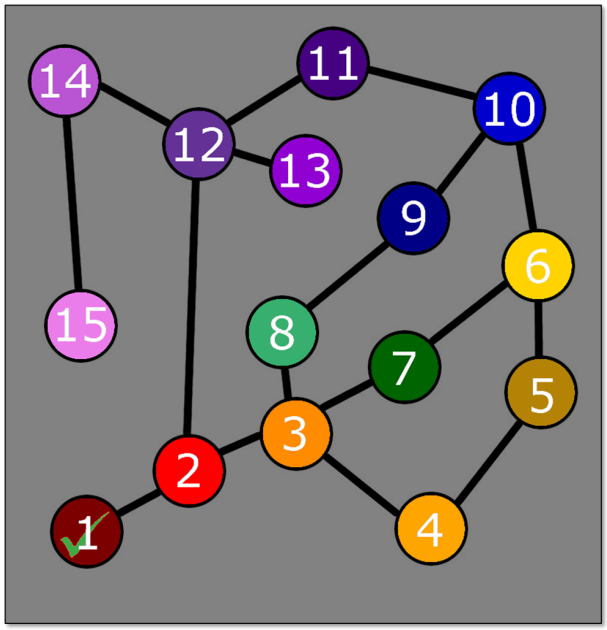

Figure 9a: Graph of the Valleys map.

Figure 9b illustrates the steady-state distribution for the player population in the role of the hider on the Valleys map. The data indicates that certain nodes exhibit a higher occupancy percentage compared to others. This allows us to draw various conclusions regarding the behavior of the hider player. The distribution suggests that a prevalent strategy involves players remaining near the initial two nodes and frequently returning to the starting node. Notably, node 9, which involves the most assets, emerges as a popular hiding spot. Nodes displaying lower values are likely to be transited by players who opt not to remain there.

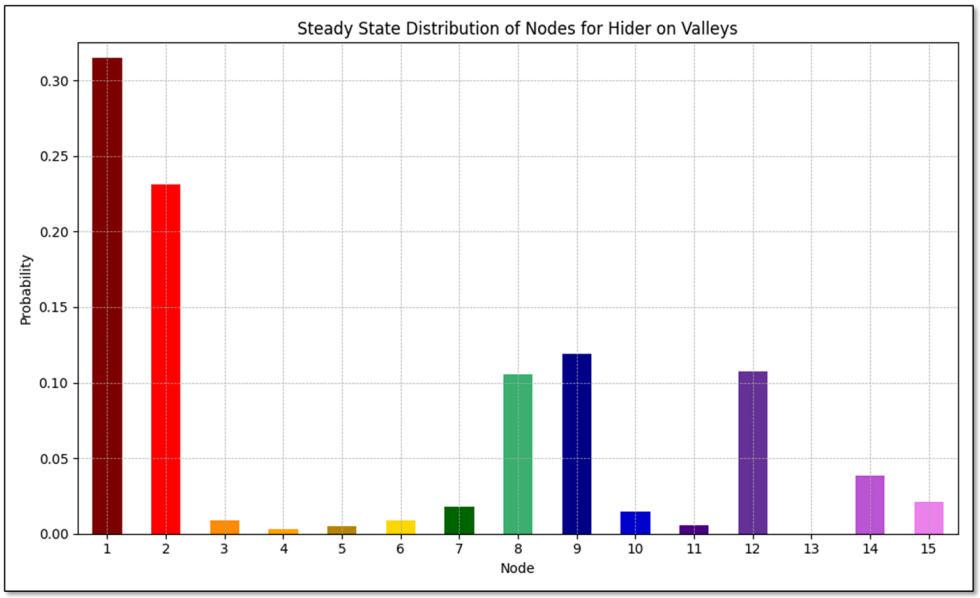

Figure 9b: Steady-state distribution of nodes for hider on Valleys.

Figure 9c illustrates the steady-state distribution for the player population assuming the role of the seeker on the Valleys map. According to the distribution, the seeker player typically inspects each node, with the time spent at each node scaling according to the number of remaining assets to be checked. This pattern is evident at node 9, where the seeker spends significantly more time thoroughly searching.

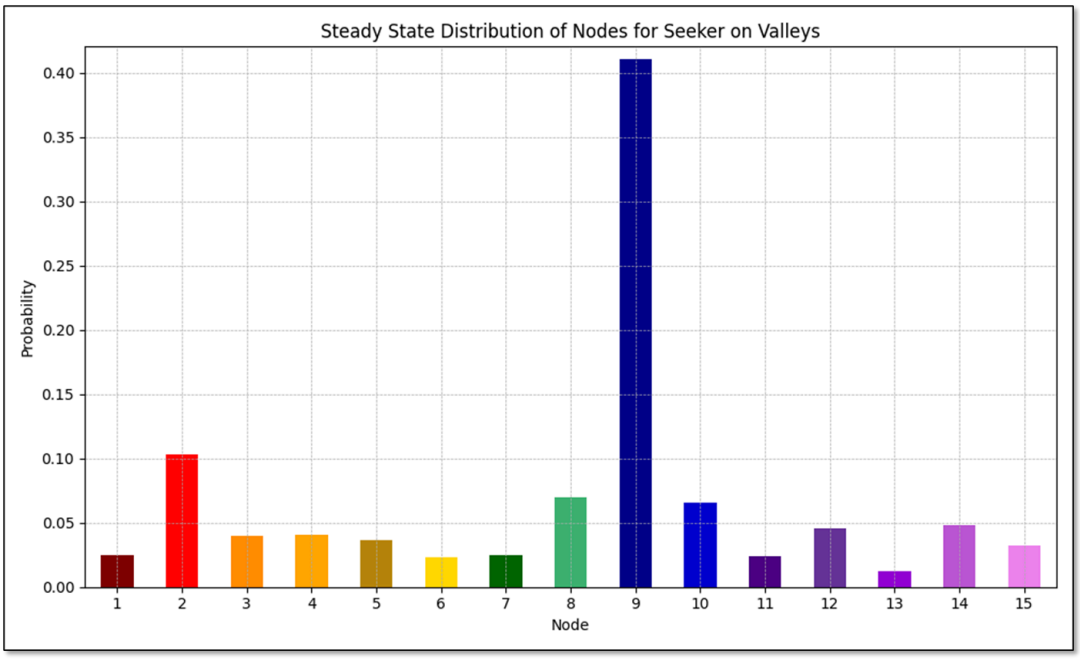

Figure 9c: Steady-state distribution of nodes for seeker on Valleys.

The data collected from the Valleys map provides valuable insights into the behaviors of both hider and seeker players. The prevailing strategy for the hider involves staying near the first two nodes. To enhance the probability of locating the hider, seekers should allocate more time to these initial nodes. Although the hider population frequently uses node 9, it is advisable for the seeker to minimize the time spent searching this node, as it does not proportionally increase their chances of success.

Graph of Garland Hall

Figure 10a shows the graph of Garland Hall, which represents the physical layout of a building at Miami University, does not conform to a traditional planar structure as its edges intersect at multiple points. As a result, it is possible to reach different areas of the map without necessarily traversing adjacent nodes. This map features more nodes and longer edges than the other maps, as a result, we can note distinct strategies adopted by the players.

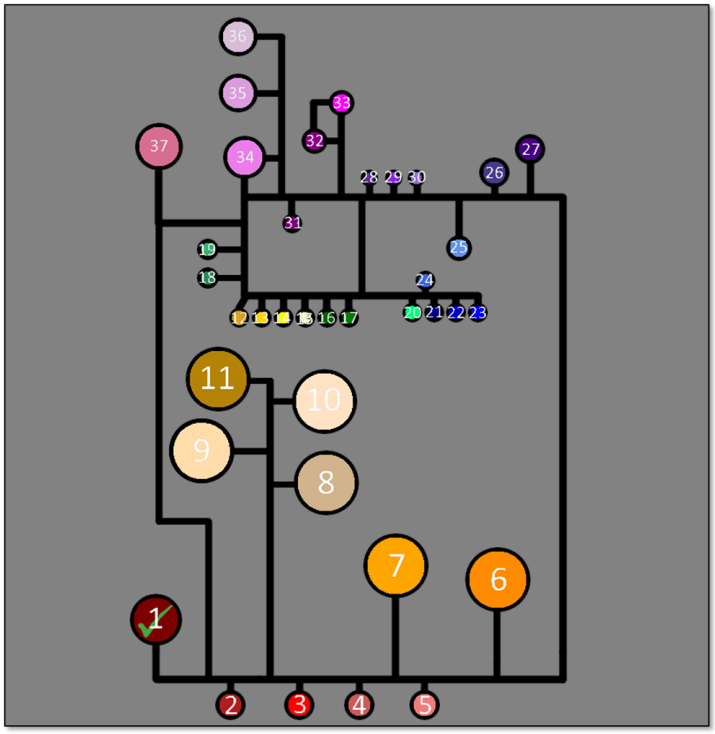

Figure 10a: Graph of Garland Hall at Miami University.

Figure 10b shows the steady-state distribution for the player population in the role of the hider on the Garland Hall map. The hider players often choose to remain stationary at a single node for the duration of the game, favoring the nodes that are furthest from the starting point. This is in contrast to the strategy observed on the Valleys map, where staying within the first few nodes is common. Interestingly, node 4 has the highest occupancy rate and suggests that the strategy of remaining close to the starting nodes is still a strategy conducted by the hider.

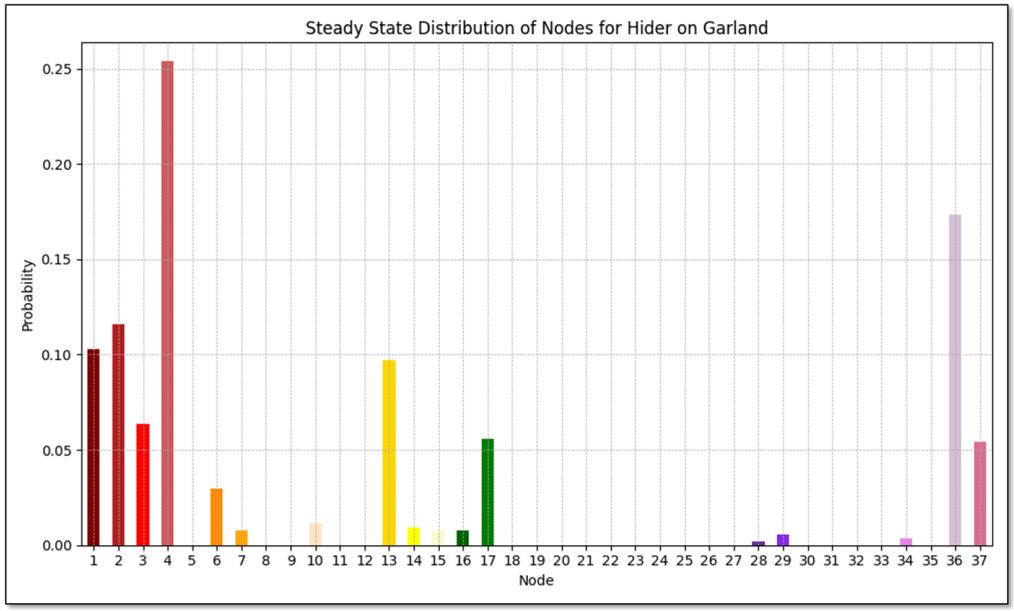

Figure 10b: Steady-state distribution of nodes for hider on Garland.

Figure 10b shows the steady-state distribution for the player population in the role of the seeker on the Garland Hall map. As Garland features less assets, the time spent at each node reflects that. The distribution is more evenly distributed compared to the other maps, with more emphasis on nodes furthest away from the starting point. Nodes 23, 36, and 37 are nodes with long distances from the start, and are evidently traversed by the seeker player population the most. The time occupied searching these nodes are directly proportional to the size of the nodes. This is because some of the nodes are small offices and others are larger classrooms.

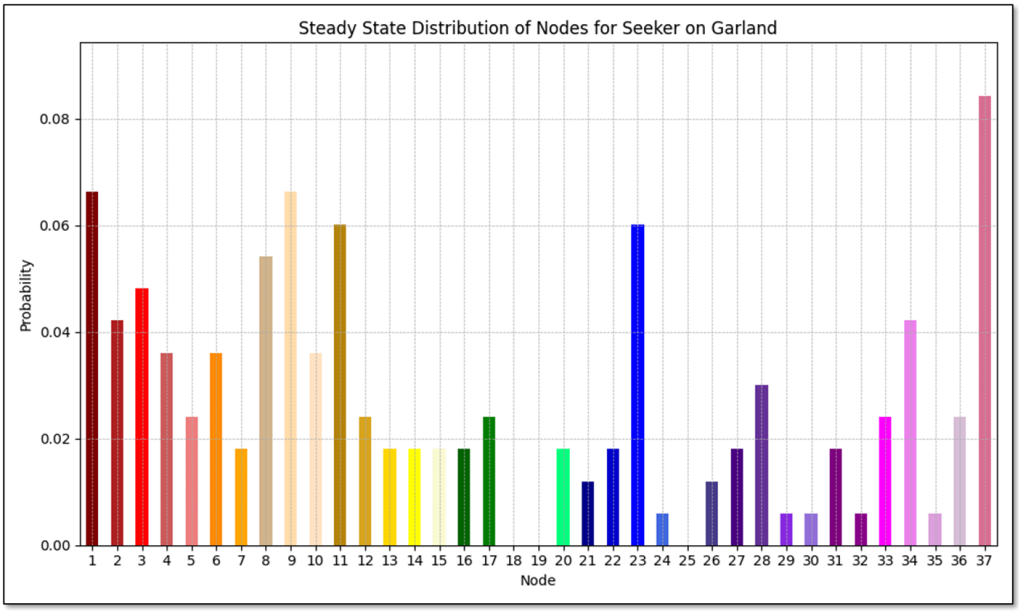

Figure 10c: Steady-state distribution of nodes for seeker on Garland.

Graph of a GPIO Circuit Diagram

Figure 11a displays the graph of a GPIO circuit. Unlike a traditional planar graph, this graph intersects at various points, albeit less frequently than the Garland Hall graph. Distinct from other maps, this layout more closely resembles a tree structure with three specific branches. The graph represents a GPIO circuit from an 8-bit AVR microcontroller, making it relevant as it models real-world physical hardware. Figure 11b provides the original diagram of the GPIO circuit. Analyzing the steady-state data from this graph could offer valuable insights into potential hotspots for hardware trojans, indicating vulnerable points in the circuit where security measures could be focused.

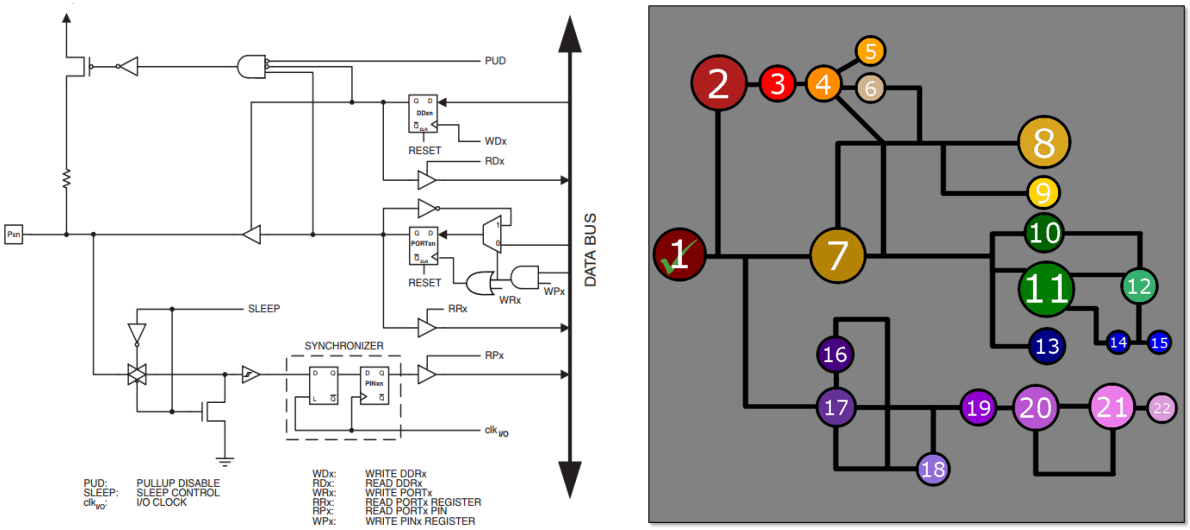

Figure 11a and 11b: Diagram and graph of a GPIO circuit from an 8-bit AVR microcontroller.

illustrates the steady-state distribution for the player population in the role of the hider on the GPIO map. The unique layout of this graph prompts hiders to select a single branch and occupy the furthest node within that branch, resulting in Nodes 8, 15, and 22 exhibiting the highest occupancy percentages. Consistent with other maps, another prevalent strategy involves retreating to the starting node as an alternative strategy.

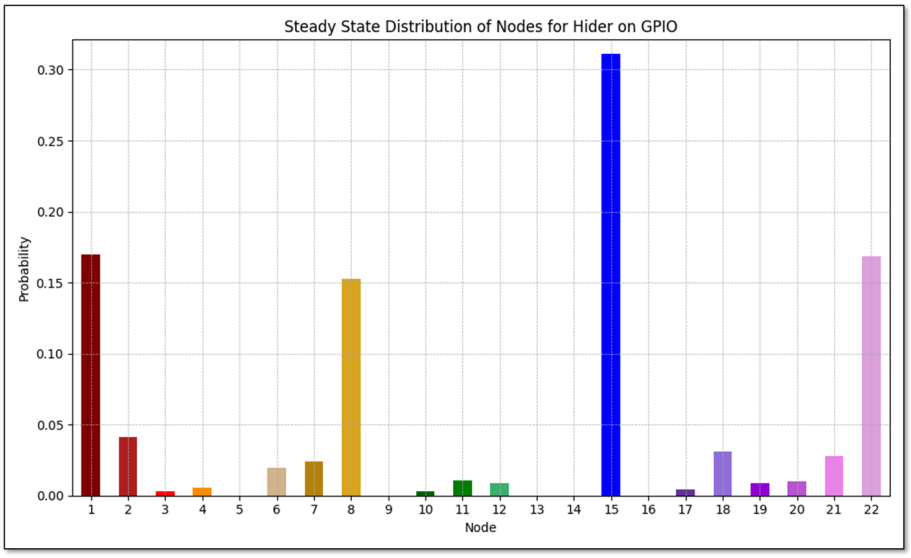

Figure 11c: Steady-state distribution of nodes for hider on GPIO.

Figure 11d depicts the steady-state distribution for the player population as seekers on the GPIO map. Given the map's tri-branch structure, the initial probability of a seeker locating a hider is approximately 33%, assuming the hider is most likely at the end of a branch. If the seeker does not find the hider in the first branch, they must then backtrack, significantly consuming time. The probability of locating the hider increases to 50% upon choosing one of the remaining two branches. This process continues until the final branch is explored, by which time the seeker may face time constraints, diminishing their chances of a successful search.

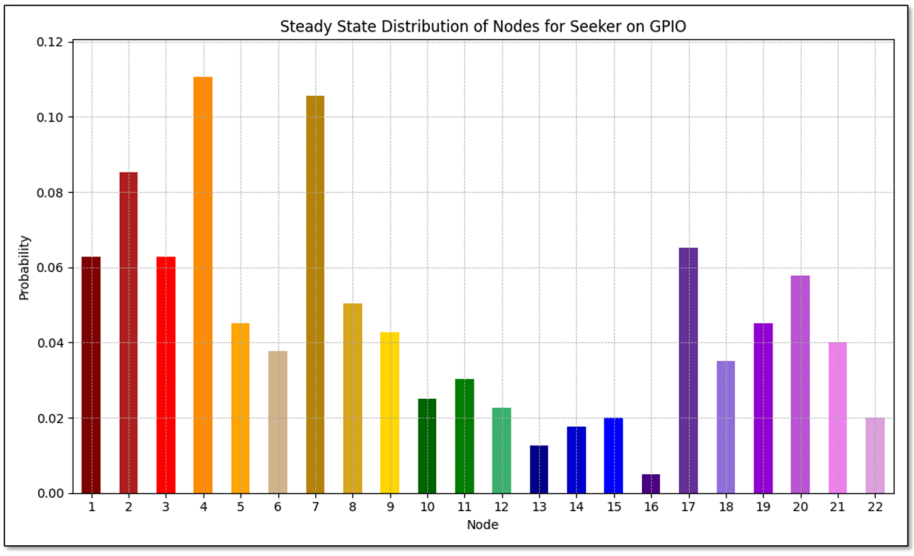

Figure 11d: Steady-state distribution of nodes for seeker on GPIO.

According to the seeker's steady-state distribution, the most advantageous strategy for the hider might involve choosing the middle branch and occupying the furthest node, as this placement maximizes the likelihood of the seeker wasting time on incorrect initial choices.

These insights into player behavior on the GPIO map provide valuable strategic understanding for both roles. The tactics observed here could be applicable in fields like cybersecurity, where understanding potential locations of threat actors is crucial. Moreover, the approach of using graph theory to map physical architectures, such as GPIO circuits, could be extended to other hardware systems and disciplines, offering broader insights into human and system behaviors.

Future Work

The project is in an early beta stage, and we have been able to get some useful data. In terms of some ideas for the future, we plan on adding some power-ups for both the hider and seeker. With the data we have collected, we plan on doing some data analysis on the player coordinate data using artificial intelligence tools like TensorFlow. When it comes to minor changes, we plan on enhancing some of the audio effects in the game. We are currently using default Unreal Engine character models which work well for our project purpose but to improve the aesthetics of the game, we plan to create our own character models and animations. We made an attempt to create our own models and animations but it proved to be too time-consuming and difficult and it provided little to no benefit to the overarching goal of our game.

Team Reflection

Throughout this project, the team worked collaboratively and efficiently within the span of the semester towards the completion of this project. Over the course of the semester, each individual displayed integrity and accountability on the completion of this project. Each team member also had a say in the decision-making process for large decisions affecting the project. The team displayed professionalism towards each other and the advisors while maintaining an open mind to constructive criticism. Time management was implemented throughout this project, and all of the tasks that were assigned have been completed. Conflicts within the team were resolved appropriately and efficiently by discussion. Overall the team gained an understanding of a professional work environment while learning how to collaborate effectively, manage time efficiently, and address challenges with maturity and resolution.

Ethical Responsibility

Globally and culturally, we strive to ensure that our approach to "The Seeker's Dilemma" is adaptable to diverse populations. We achieve this goal by considering local customs and accessibility needs, making sure it is inclusive and culturally acceptable. Our game has been

developed with a comprehensive understanding of global, cultural, social, environmental, and economic factors. Our commitment to ethical considerations extends beyond cultural and societal aspects to encompass privacy and data protection. In our Hide&Seek game, we prioritize the confidentiality and security of the data collected during gameplay.

Health, Safety, and Welfare

As engineers, it is crucial to consider the potential impacts our game may have on public health, safety, and welfare. We assessed the potential risks associated with our game, ensuring that they do not pose threats to human health or safety. Additionally, our game aims to improve the overall quality of life and contribute positively to societal welfare by providing a fun and educational game used for solving “The Seeker’s Dilemma”. Environmentally, we prioritize digital distribution to reduce physical waste. Economically, we aim to make the game accessible across diverse economic backgrounds, thereby enhancing economic inclusivity.

Conclusion

In this project, we successfully developed a Hide&Seek game using Unreal Engine as a simulation to illustrate key concepts in the cybersecurity and hardware trojan space. Our game effectively emulates the seeker’s dilemma, demonstrating exploration, exploitation, and analysis of human behavior in a simulated environment.

Our choice of Unreal Engine 5.2.1 proved to be the correct choice. Despite initial challenges in adapting to different engine versions, our progress in learning how to effectively use C++ and blueprints facilitated our development progress. We migrated from Epic Online Services to Steamworks for our client-server model to provide an excellent, user-friendly framework for our multiplayer setup.

The gameplay framework we developed allowed for a truly modular and scalable game design. We scaled from one map to four, each with different environments, hiding places, and a real-world application. Maps can be designed and implemented with ease because of an improved design implementation. Using states gave us a structured approach to organization and made sense when developing the game. Our heatmap feature offers insight into player behaviors and strategies, mirroring the analytical aspect of the cybersecurity space. This data can be graphed and analyzed to understand how humans think and strategize Hide&Seek.

The analysis of the data was done with Python. A MongoDB database is used to store the data. The Python script translates the raw data into heatmaps to visualize where the hiders and seekers traveled and popular hiding spots. A Markov Chain model and a steady state distribution is the mathematical approach to the data to better understand the popular strategies of hide and seek.

We experienced many challenges throughout the project. Replication and the low-level multiplayer approach were a few of these challenges. Multiplayer issues required a complete reconstruction of our system. However, our team's adaptability as engineers created innovative solutions to overcome these challenges.

In conclusion, this project not only achieved its aim of creating an engaging, functional, and fun Hide&Seek game but also served as a practical metaphor for cybersecurity concepts. We maintained a commitment to ethical engineering practices. Additionally, our game design respected privacy, inclusivity, and data security. We hope that the insights gained from this project will allow us to link hardware trojan detection to game theory, and truly understand the concept of the Seeker's Dilemma.

Citations

[1] The Seeker's Dilemma paper, provided by Dr. Peter Jamieson

a. **Citation**: *The Seeker's Dilemma: Formulation and Benchmarking of Hardware Trojan Detection with Game Theory*.

[2] Unreal Engine 5.2.1

a. **Citation**: Epic Games, 2019. Unreal Engine, Available at: https://www.unrealengine.com.

[3] Epic Games Developer Portal

a. **Citation**: *Epic Developer Portal*, dev.epicgames.com/portal/en-US/epic-games. Accessed 2023.

[4] Quixel Bridge, assets used in map design

a. **Citation**: "Quixel Bridge - Manage 3D Content and Export with One Click." *Quixel*, quixel.com/bridge. Accessed 2023.

[5] EOS multiplayer setup

a. **Citation**: "Epic Online Services #1 - Setup Plugin, Login, Getter Functions - C++/Blueprints - Unreal Engine 5." *YouTube*, YouTube, 17 Nov. 2022, www.youtube.com/watch?v=KvyXSJ5vG3U&list=PL0jFyH3meZDNDcOMqs1UIMlB1jhTCW_pj.

b. **Citation:** "Epic Online Services #2 - Create/Find/Join Sessions - C++/Blueprints - Unreal Engine 5." YouTube, YouTube, 18 Nov. 2022, www.youtube.com/watch?v=Ek5XetMzI64&list=PL0jFyH3meZDNDcOMqs1UIMlB1jhTCW_pj&index=2.

[6] Flashlight implementation

a. **Citation**: "How to Make a Flashlight in Unreal Engine 5." *YouTube*, YouTube, 21 Jan. 2023, www.youtube.com/watch?v=LQ3NzUL4ioE.

[7] Flashlight replication implementation

a. **Citation:** "Replication | First Person Flashlight - Unreal Engine Tutorial." *YouTube*, YouTube, 3 July 2021, www.youtube.com/watch?v=nS8d103CTJM.

[8] Role selection with hider and seeker

a. **Citation**: "Team System Unreal Engine 5 | How to Make a Multiplayer Team System in Unreal Engine 5?" *YouTube*, YouTube, 30 Jan. 2023, www.youtube.com/watch?v=rvFbHhysCu4.

[9] Steam UI, lobby, friendlist, and character selection

a. **Citation**: "How to Add Steam Multiplayer in Unreal Engine 5 - Tutorial." *YouTube*, YouTube, 12 Nov. 2023, www.youtube.com/watch?v=E2okoXqsXNc.
b. **Citation**: "Steam Lobby / Party - Part 1/4 - Unreal Engine 5 Tutorial [Ue5]." *YouTube*, YouTube, 5 Sept. 2023, www.youtube.com/watch?v=NUKTKFYbW_w.
c. **Citation**: "Steam Lobby / Party - Part 2/4 - Unreal Engine 5 Tutorial [Ue5]." *YouTube*, YouTube, 6 Sept. 2023, www.youtube.com/watch?v=jpeofkeVXSA.
d. **Citation**: "Steam Lobby / Party - Part 3/4 - Unreal Engine 5 Tutorial [Ue5]." *YouTube*, YouTube, 8 Sept. 2023, www.youtube.com/watch?v=OqetLmStZiA.
e. **Citation**: "Steam Lobby / Party - Part 4/4 - Unreal Engine 5 Tutorial [Ue5]." *YouTube*, YouTube, 13 Sept. 2023, www.youtube.com/watch?v=e3yKqhG3C5g.
f. **Citation**: "Steam Friends List & Invite Player - Unreal Engine 5 Tutorial [Ue5]." *YouTube*, YouTube, 14 Aug. 2023, www.youtube.com/watch?v=ya6FQktr_OE.
g. **Citation**: "Character Selection in Steam Lobby - Unreal Engine 5 Tutorial [Ue5]." *YouTube*, YouTube, 16 Oct. 2023, www.youtube.com/watch?v=V_959mB16OQ.

[10] Background on Hide&Seek on a graph

a. **Citation:** "M. Chapman, G. Tyson, P. McBurney, M. Luck, and S. Parsons, 'Playing hide-and-seek: an abstract game for cyber security,' in Proceedings of the 1st International Workshop on Agents and CyberSecurity, 2014, pp. 1–8."