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

Fall 2023/Spring 2024

College of Engineering and Computing

Department of Electrical and Computer Engineering

ECE 448: Senior Design

Advised by Dr. Jamieson and Dr. Bhunia

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

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

Jared Butler butler57@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-and-seek game as a metaphorical simulation for cybersecurity practices. Hide-and-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-and-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 are building a two-player online hide-and-seek game. While playing the game, the positions of both players are constantly being recorded. A heatmap is continuously being 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

Background

The background of the project revolves around the exploration of the dynamics of a hide-and-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-and-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-and-seek game align with the principles. The design and implementation of our game are 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)

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 current server-client implementation utilizes EOS. The reason for choosing EOS 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.

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.

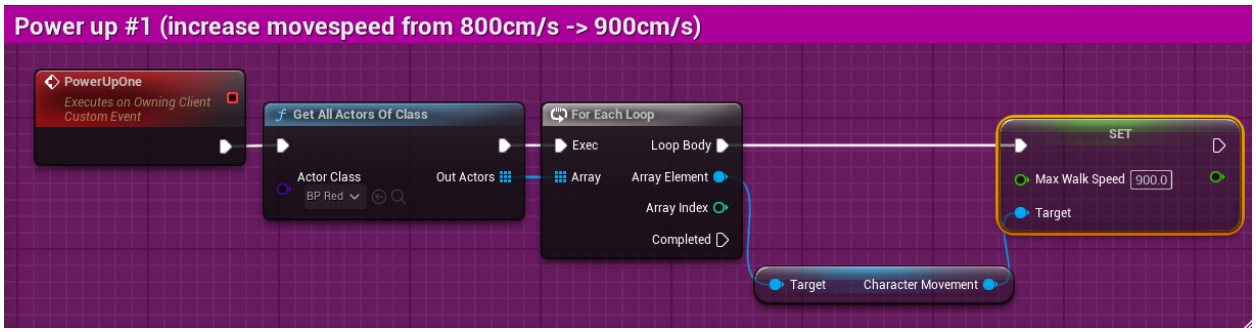

Figure 1: Blueprint visual script for a single power-up.

The figure has 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. Seeker power-ups are also defined in this class. Refer to Figure 1 to see the implementation of this power-up. 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 server and clients. We utilize this class to produce player coordinate arrays that are used in producing our hider and seeker heatmaps.

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. Starting and ending points are identified by green and red dots respectively. 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 an example heatmap of the seeker player during a short gameplay test. The heatmap uses 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.

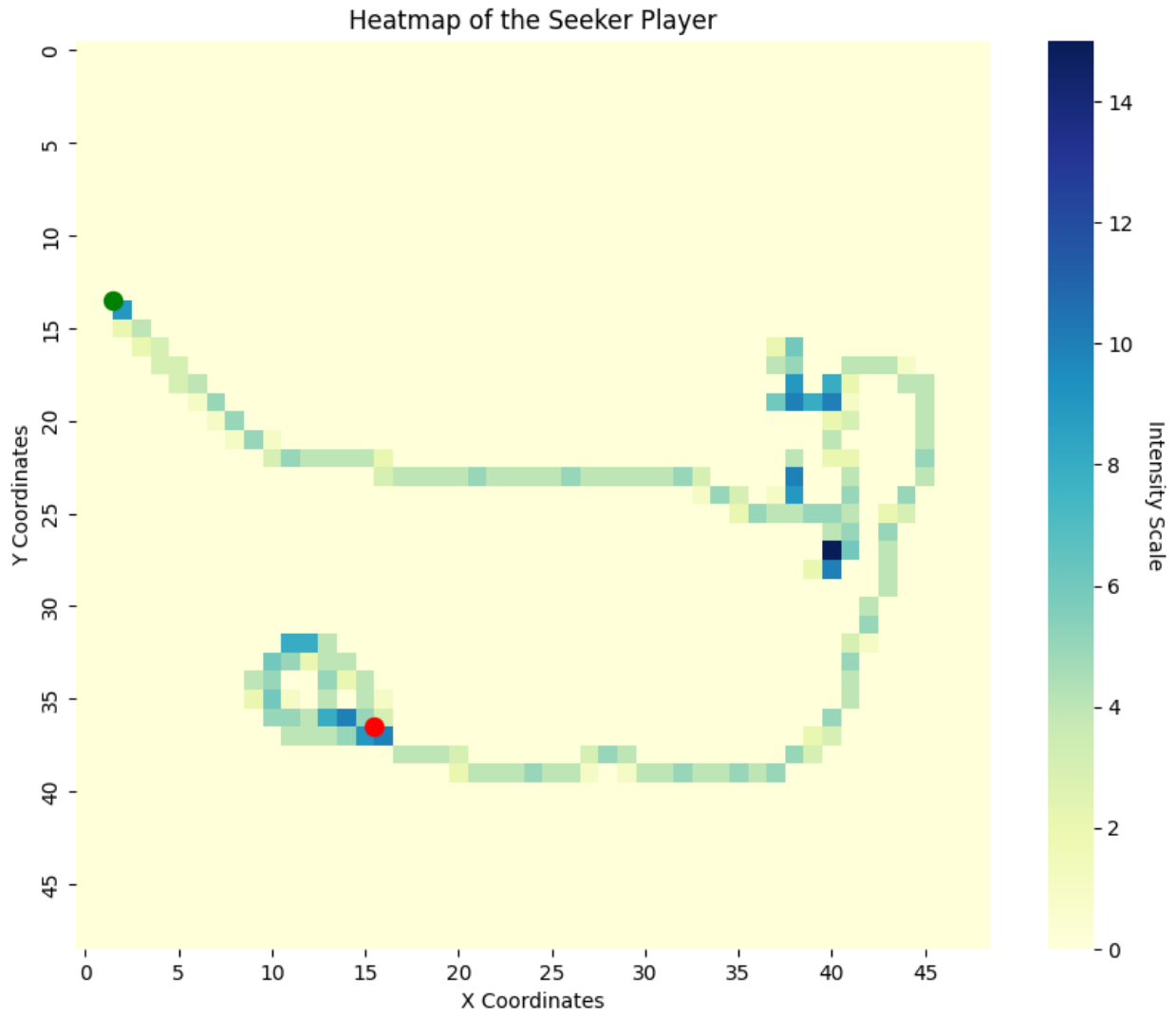

Figure 2: Heatmap of the seeker player.

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, the countdown timer for the game to start, and UI design to allow a player to choose between the role of a hider or a seeker.

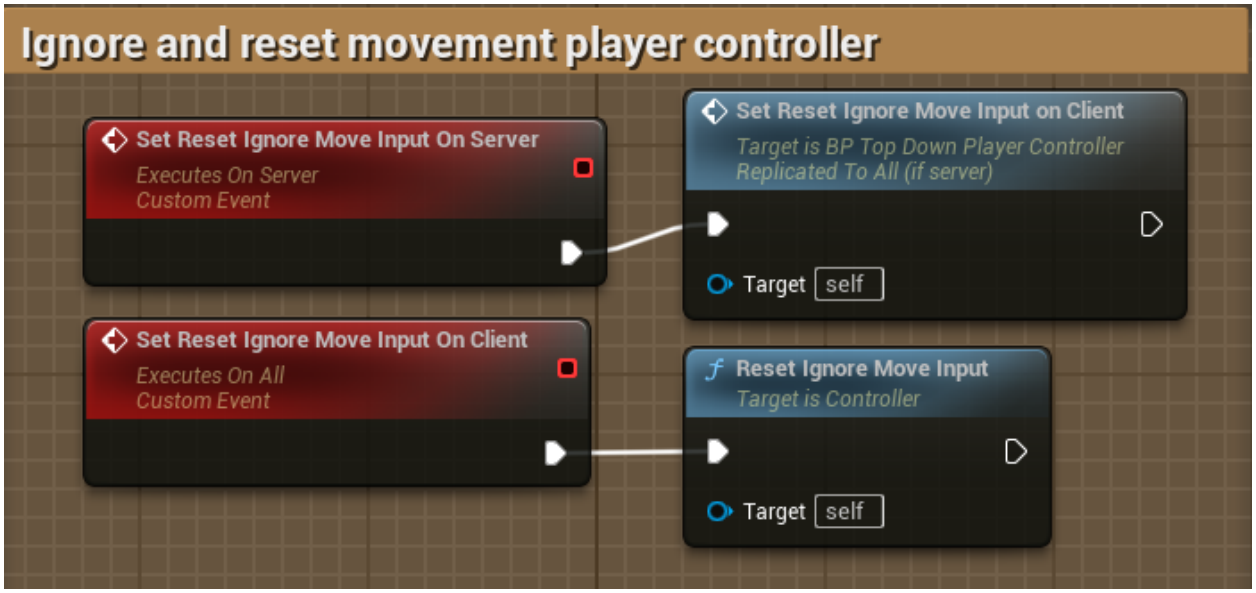

Figure 3: Blueprint representation of RPC.

Low-Level Multiplayer Approach

We first tried to implement multiplayer at a higher level, within the blueprints themselves. This became problematic as our version of Unreal Engine was too new and required new methods of implementation. While creating a session worked well on the server side, joining from the client side was an issue. We decided to pivot to a lower-level approach for dealing with multiplayer sessions. We coded the 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.

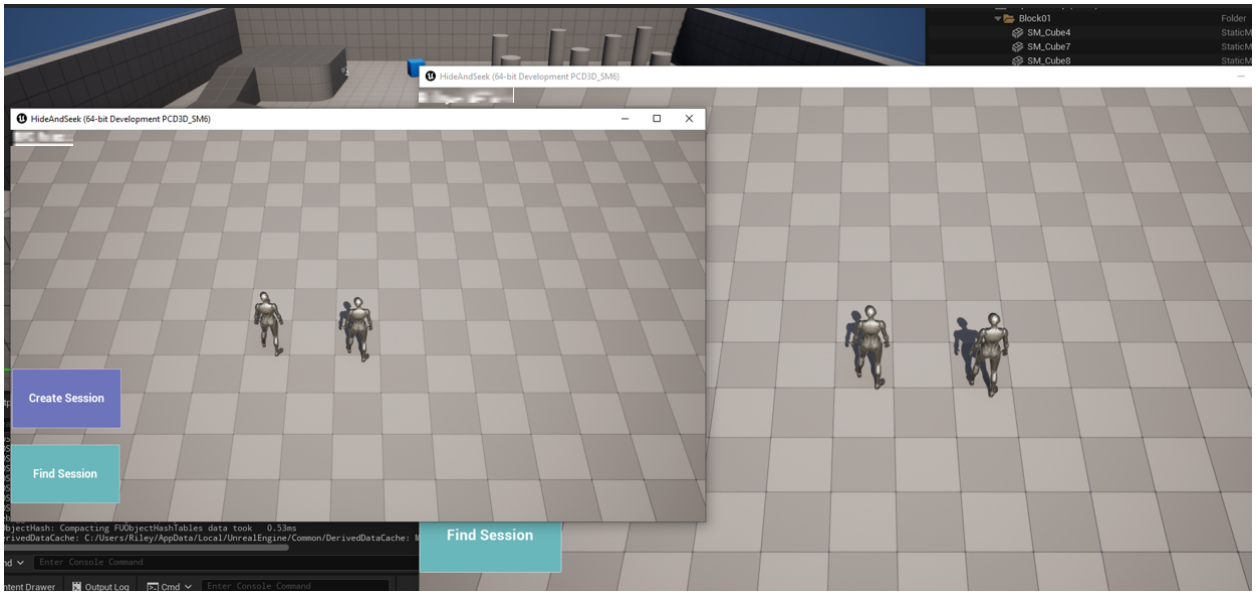

Figure 4: Successful connection with the server and client model.

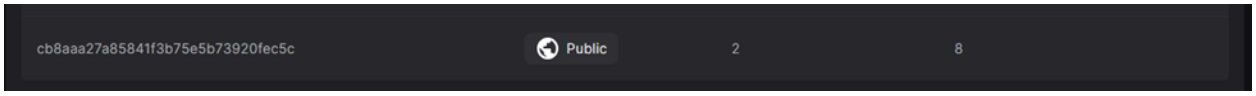

Figure 5: Developer portal server with connections shown.

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 is a two-player, top-down, third-person view hide-and-seek simulator. 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 log in and connect to sessions. Figure 6 shows the main menu screen where the user can log in and then create or find sessions.

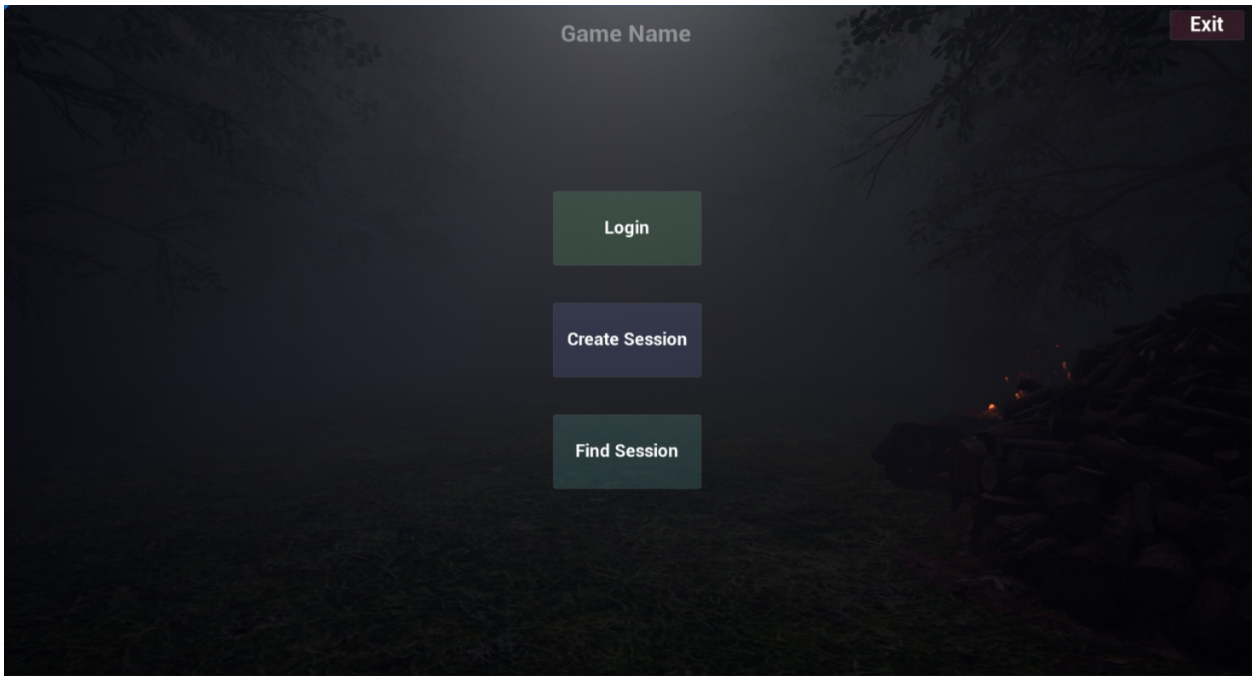

Figure 6: Main menu screen featuring multiplayer functionality.

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. Figures 7 and 8 show role selection once the game begins and how the seeker can interact with the hider, respectively. These figures also display an information bubble. Once hovered, it will explain the controls of the game. Finally, We decided to set the game to nighttime, adding a flashlight as an additional strategic element for each of the players.

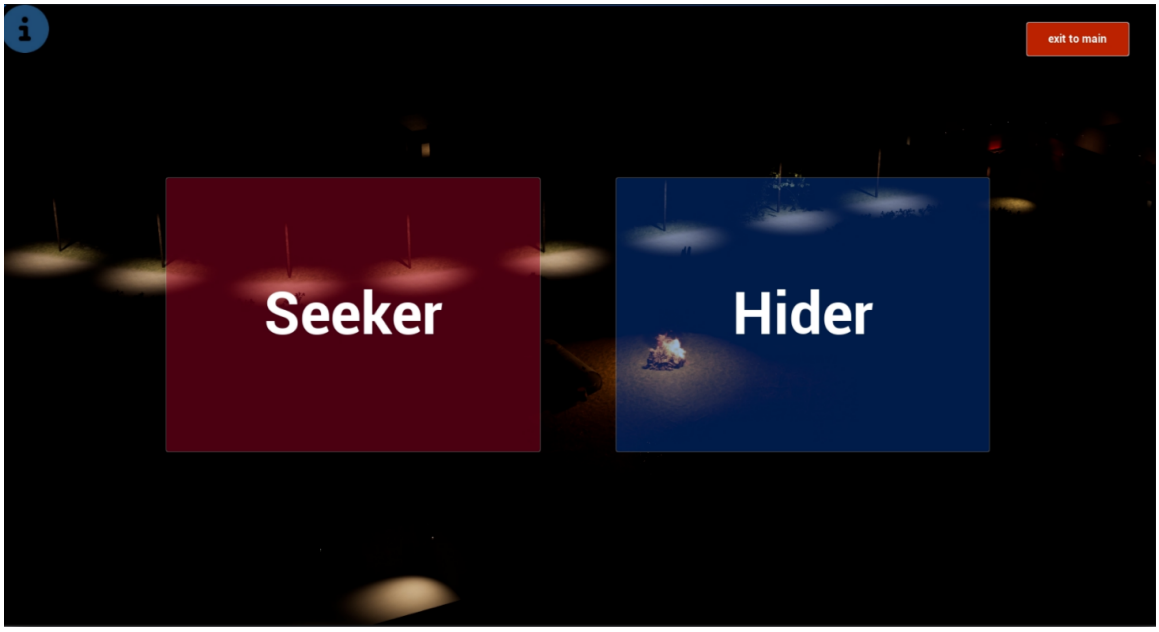

Figure 7: Role selection screen after creating a session.

Figure 8: Seeker interaction by proximity.

Late Development Stage

Once we finished how the game is played, we designed a single map with easy-to-identify static landmarks that will be used for analyzing the generated heatmap after the game. The assets used in the map are from Quixel Bridge, a desktop application that serves as a supporting tool for downloading, importing, and exporting Megascans assets. Using assets and plugins included in UE5, the map design is straightforward. The maps include a mansion, a small house, two shacks, a lake house, a lake, foliage, small areas for campfires, and artificial and natural boundaries in the map. Obtaining an engaging, high-fidelity map is essential for quality gameplay and for establishing landmarks for analyzing player data. After the map is designed, we can start player testing and adding features.

Figure 9: Completed map with unique landmarks.

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.

Future Work

We plan on making some more changes to the game. But before we make these changes, we need to conduct some individual player testing to see how our game is perceived. After we get a consensus, we plan on adding some more power-ups for both the hider and seeker. We also 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 adding some audio effects to the game, but we are wondering if that could change the dynamics of the game. For example, if the hider hears the seeker approaching, would they still stay in the same position or will they be more inclined to find a new hiding position?

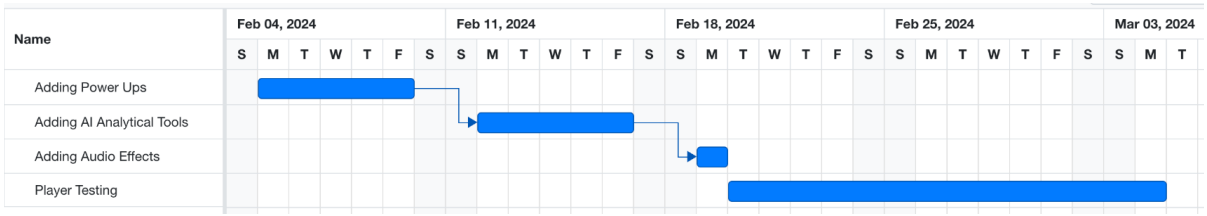

Figure 10: Gannt chart of future work to be completed.

Team Ethics

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 tasks assigned were 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.

Conclusion

In this project, we successfully developed a hide-and-seek game using Unreal Engine as a simulation to illustrate key concepts in the cybersecurity space. Our game effectively emulates the seeker's dilemma, demonstrating exploration and exploitation.

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. The use of Epic Online Services for our client-server model streamlined the integration and provided an excellent framework for our multiplayer setup.

The gameplay framework we developed allowed for a truly modular and scalable game design. The use of 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.

We experienced many challenges throughout the project. Replication and the low-level multiplayer approach were a few of these challenges. 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 and functional hide-and-seek game but also served as a practical metaphor for cybersecurity concepts. We hope that 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) Megascan trees, for the larger trees in the map
    a. **Citation**: "Get Started with Megascans Trees." *YouTube*, YouTube, 15 Dec. 2021, www.youtube.com/watch?v=-QK5gaeEAig.