Mya Souvanna

Dr. Basu

CPSC 3384

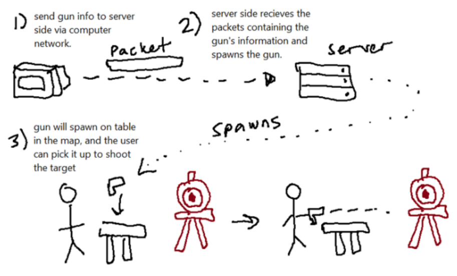
03 May 2023

Final Project Report

Introduction

Ever since I was a child, games have been a huge part of my life. I wanted to create a game-inspired final project since that's a huge part of my life, and I believe that one would better enjoy making a project if it includes things that they're interested in. As I decided on what I was going to create, I planned on going with a project that involved games in some way, shape, or form. Initially, I was planning to do a project revolving around a shooting range. A random gun would spawn into the scene based on the server's message, and the player would then shoot a target. However, I decided to slightly alter some details of my initial project and to have a project that'd only have a player (a capsule), a plane, and a cube (used to show the initial position of the player).

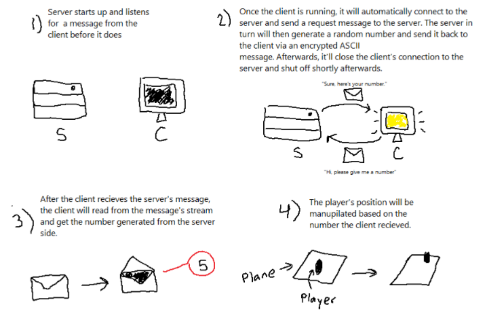
Original Project: "Shooting Range"


1) send gun info to server side via computer network.

2) server side recieves the packets containing the gun's information and spawns the gun.

3) gun will spawn on table in the map, and the user can pick it up to shoot the target

Final Project: "Random Positions"


1) Server starts up and listens for a message from the client before it does

2) Once the client is running, it will automatically connect to the server and send a request message to the server. The server in turn will then generate a random number and send it back to the client via an encrypted ASCII message. Afterwards, it'll close the client's connection to the server and shut off shortly afterwards.

3) After the client recieves the server's message, the client will read from the message's stream and get the number generated from the server side.

4) The player's position will be manupilated based on the number the client recieved.

Methodology

After the vision documents were drafted, I spent some time doing research on how I could make this project a real thing. Some of the tools that I had used (but not limited to) included the Zoom class recordings, the class Google Drive, and YouTube. Whenever I felt like I had gathered and researched enough information, I began the actual coding. As I was coding within Unity, I realized that my ParrelSync wasn't working because I wasn't able to download it to my Project Manager via its GitHub URL. This was the first (of many) obstacles that I had encountered while making this project. Instead of spending time on trying to figure out how to make it work, I decided to improvise and think outside the box. Instead of using ParrelSync, I opted for using a console application for the server-side of my project after doing research on it.

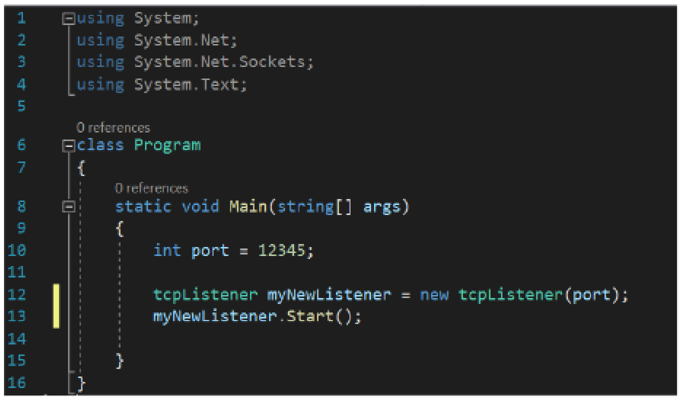
As a few weeks passed, I realized that I may have been a little too ambitious for my project so I quickly changed it from the "Shooting Range" project to the "Random Positions" project. Based on a random number generated from the server, I decided that instead of spawning something in, I'd use it to change the player's position on the plane with the use of a random number generator.

Code Snippets

Below are screenshots of my code snippets. On the server-side of my code, it mainly created a new TCP listener and waited patiently for a request from the client. After it received the client's request, it'd randomly generate a number and then send it back to the client. On the client-side of my code, the client would send a request to the server for a random number. After it received the number from the server, it used that number to change the position of the player (via transform.position) and changed the x and z coordinates of the player. It also changed the color of the player if they clicked on the player and sent another request to the server for another randomly generated number.

Server-side code:

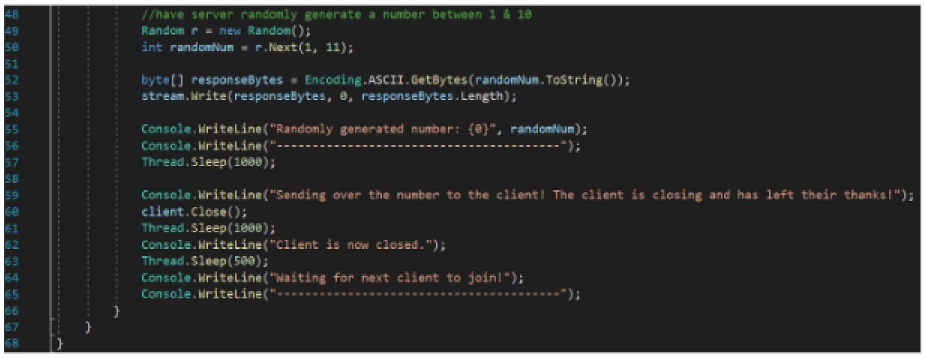
Server-side: program.cs



```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        int port = 12345;

        tcpListener myNewListener = new tcpListener(port);
        myNewListener.Start();

    }
}
```

- Initialized my port to '12345'
- I assign 'myNewListener' as an instance of my new tcpListener class. It takes in 'port' as a parameter since it'll be the port that the server listens on
- myNewListener.Start() will begin listening for incoming connections to the server on the port number .

Server-side: tcpListener.cs Start() 2

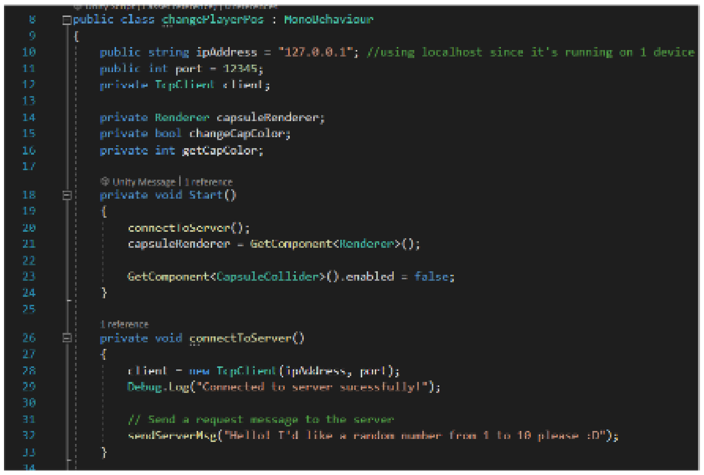


```
            //have server randomly generate a number between 1 & 10
            Random r = new Random();
            int randomNum = r.Next(1, 11);

            byte[] responseBytes = Encoding.ASCII.GetBytes(randomNum.ToString());
            stream.Write(responseBytes, 0, responseBytes.Length);

            Console.WriteLine("Randomly generated number: {0}", randomNum);
            Console.WriteLine("----------------------------------------");
            Thread.Sleep(1000);

            Console.WriteLine("Sending over the number to the client! The client is closing and has left their thanks!");
            client.Close();
            Thread.Sleep(1000);
            Console.WriteLine("Client is now closed.");
            Thread.Sleep(500);
            Console.WriteLine("Waiting for next client to join!");
            Console.WriteLine("----------------------------------------");
        }
    }
}
```

- Once the client's message is received, a random number is then generated from a scale of 1 to 10.
- Afterwards, the server sends the number back over to the client, closes the client, and waits for the next client to join.

Client-side code:

Client-side: changePlayerPos.cs 1

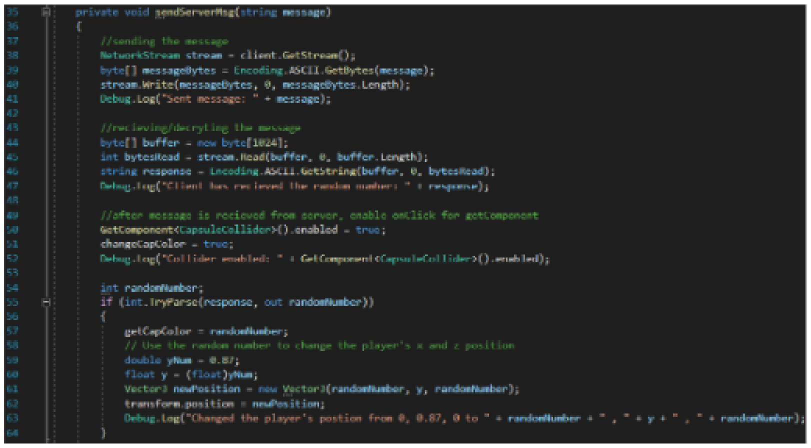


```
public class changePlayerPos : MonoBehaviour
{
    public string ipAddress = "127.0.0.1"; //using localhost since it's running on 1 device
    public int port = 12345;
    private TcpClient client;

    private Renderer capsuleRenderer;
    private bool changeCapColor;
    private int getCapColor;

    Unity Message | 1 reference
    private void Start()
    {
        connectToServer();
        capsuleRenderer = GetComponent<Renderer>();

        GetComponent<CapsuleCollider>().enabled = false;
    }

    1 reference
    private void connectToServer()
    {
        client = new TcpClient(ipAddress, port);
        Debug.Log("Connected to server sucessfully!");

        // Send a request message to the server
        sendServerMsg("Hello! I'd like a random number from 1 to 10 please :D");
    }
```

CPSC 3384 - Computer Networks - Spring 2023

- IPAddress and port is initalized on the client side. These are used to help establish a connection to the server.
    - capsuleRenderer, changeCapColor, and getCapColor is used to change the player's color since a message is received from the server
- In Start(), the connectToServer() is called.
    - Similar to listener in tcpListener.cs, client connects to the server with the given IP address and port.
    - sendServerMsg is also called to send a specified message to the server

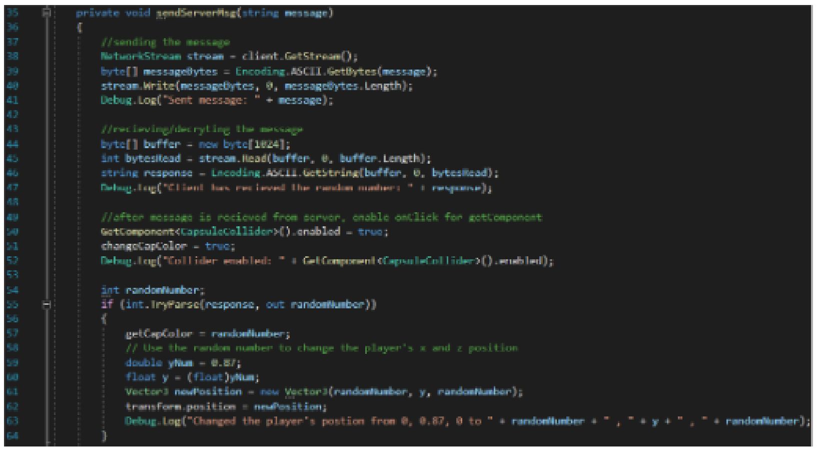
Client-side: changePlayerPos.cs sendServerMsg() 1



```
private void sendServerMsg(string message)
{
    //sending the message
    NetworkStream stream = client.GetStream();
    byte[] messageBytes = Encoding.ASCII.GetBytes(message);
    stream.Write(messageBytes, 0, messageBytes.Length);
    Debug.Log("Sent message: " + message);

    //recieving/decryting the message
    byte[] buffer = new byte[1024];
    int bytesRead = stream.Read(buffer, 0, buffer.Length);
    string response = Encoding.ASCII.GetString(buffer, 0, bytesRead);
    Debug.Log("Client has recieved the random number: " + response);

    //after message is recieved from server, enable onClick for getComponent
    GetComponent<CapsuleCollider>().enabled = true;
    changeCapColor = true;
    Debug.Log("Collider enabled: " + GetComponent<CapsuleCollider>().enabled);

    int randomNumber;
    if (int.TryParse(response, out randomNumber))
    {
        getCapColor = randomNumber;
        // Use the random number to change the player's x and z position
        double yNum = 0.87;
        float y = (float)yNum;
        Vector3 newPosition = new Vector3(randomNumber, y, randomNumber);
        transform.position = newPosition;
        Debug.Log("Changed the player's postion from 0, 0.87, 0 to " + randomNumber + " , " + y + " , " + randomNumber);
    }
```

- sendServerMsg() is used to send and receive messages from the server. Also is used to change the player's position.
    - To send the message to the server, it will convert the message into a bytes array and then encode it via ASCII.
    - Using stream.read, it reads the response and decodes it back into a string.

Client-side: changePlayerPos.cs sendServerMsg() 2

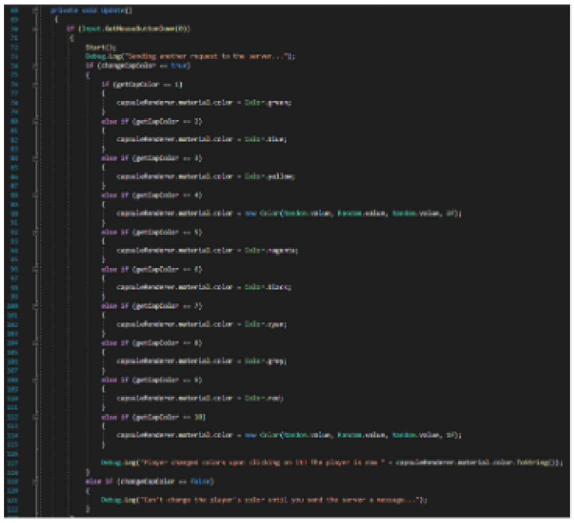


```
private void sendServerMsg(string message)
{
    //sending the message
    NetworkStream stream = client.GetStream();
    byte[] messageBytes = Encoding.ASCII.GetBytes(message);
    stream.Write(messageBytes, 0, messageBytes.Length);
    Debug.Log("Sent message: " + message);

    //recieving/decryting the message
    byte[] buffer = new byte[1024];
    int bytesRead = stream.Read(buffer, 0, buffer.Length);
    string response = Encoding.ASCII.GetString(buffer, 0, bytesRead);
    Debug.Log("Client has recieved the random number: " + response);

    //after message is recieved from server, enable onClick for getComponent
    GetComponent<CapsuleCollider>().enabled = true;
    changeCapColor = true;
    Debug.Log("Collider enabled: " + GetComponent<CapsuleCollider>().enabled);

    int randomNumber;
    if (int.TryParse(response, out randomNumber))
    {
        getCapColor = randomNumber;
        // Use the random number to change the player's x and z position
        double yNum = 0.87;
        float y = (float)yNum;
        Vector3 newPosition = new Vector3(randomNumber, y, randomNumber);
        transform.position = newPosition;
        Debug.Log("Changed the player's postion from 0, 0.87, 0 to " + randomNumber + " , " + y + " , " + randomNumber);
    }
```

- Once the message from the server is decoded to a string, capsuleCollider and changeColor is set to true (allowing the user to press the player).
- getCapColor is assigned the randomNumber value.
    - Used to determine what color the capsule will be once user clicks it.
- randomNumber is also used to change the player's position from (0, 0.87, 0) to (randomNumber, 0.87, randomNumber).

Client-side: changePlayerPos.cs Update()

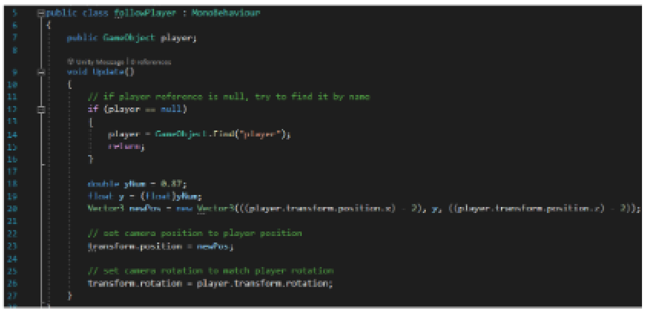
- If a message had been received by the server and changeCapColor = true, then the client will send out another request to the client via Start().
    - Also will change the color of the capsule depending on the randomNumber that was generated by the server.
- Else, if the client hadn't initially sent a message to the client and hadn't received a message back from the server either, a message will appear stating that the color won't change until a message is sent to the server



```csharp
private void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Start();
        Debug.Log("Sending another request to the server...");
        if (changeCapColor == true)
        {
            if (getCapColor == 1)
            {
                capsuleRenderer.material.color = Color.green;
            }
            else if (getCapColor == 2)
            {
                capsuleRenderer.material.color = Color.blue;
            }
            else if (getCapColor == 3)
            {
                capsuleRenderer.material.color = Color.yellow;
            }
            else if (getCapColor == 4)
            {
                capsuleRenderer.material.color = new Color(Random.value, Random.value, Random.value, 1f);
            }
            else if (getCapColor == 5)
            {
                capsuleRenderer.material.color = Color.magenta;
            }
            else if (getCapColor == 6)
            {
                capsuleRenderer.material.color = Color.black;
            }
            else if (getCapColor == 7)
            {
                capsuleRenderer.material.color = Color.cyan;
            }
            else if (getCapColor == 8)
            {
                capsuleRenderer.material.color = Color.grey;
            }
            else if (getCapColor == 9)
            {
                capsuleRenderer.material.color = Color.red;
            }
            else if (getCapColor == 10)
            {
                capsuleRenderer.material.color = new Color(Random.value, Random.value, Random.value, 1f);
            }

            Debug.Log("Player changed colors upon clicking on it! The player is now " + capsuleRenderer.material.color.ToString());
        }
        else if (changeCapColor == false)
        {
            Debug.Log("Can't change the player's color until you send the server a message...");
        }
```

Client-side: followPlayer.cs

- Simple code to have the mainCamera of my scene follow the player.
    - Used so that the user can easily click on the capsule to change its color



```csharp
public class followPlayer : MonoBehaviour
{
    public GameObject player;

    Unity Message | 0 references
    void Update()
    {
        // if player reference is null, try to find it by name
        if (player == null)
        {
            player = GameObject.Find("player");
            return;
        }

        double yNum = 0.87;
        float y = (float)yNum;
        Vector3 newPos = new Vector3(((player.transform.position.x) - 2), y, ((player.transform.position.z) - 2));

        // set camera position to player position
        transform.position = newPos;

        // set camera rotation to match player rotation
        transform.rotation = player.transform.rotation;
    }
```

Challenges of the Project

Like probably everyone else working on this project, I had run into challenges here and there that slowed me down. There were mainly 2 challenges that I had run into: ParrelSync and Missing Models/Assets. Because I couldn't get ParrelSync to download to my ProjectManager, I had to rethink how I was going to set up my server-client architecture; I solved this by doing research and finding an alternative method [Console Application]. Aside from that, I realized that in order to create a "Shooting Range" game, I needed models for both the guns and targets.

As I was researching gun models and such, I realized that I wanted to spend more time on coding the server-client architecture and not focus so much on the design of my project. So, in order to fix this problem, I had decided to change the entire basis of my project from generating an object into the project, I would instead just change the position of a gameObject within my project.

The 5 Challenges of Computer Network: How Is It Present & How Can I Fix It?

From the 7 challenges that we discussed in class, I found that my project had these challenges in it including Project Degradation, Security Issues, Slow Connectivity, Capacity Concerns, and Monitoring & Maintenance.

Project Degradation:

How was it present within my project?

In the server-side code, I repeatedly used Thread.Sleep() so that the consolelog wouldn't spew the messages all at once. This causes poor performance since it forces the current thread to pause for a certain amount of time, which in turn leads to wasted CPU use and cycles. If thread.sleep() goes on for too long, the entire thread is idle for that said time and it wastes CPU cycles that could've been used for other tasks. Else, if thread.sleep() is too short, it could cause overheads that can build up overtime

How can I fix it?

To fix project degradation that forms because of my constant use of thread.sleep(), I could use alternative methods that essentially do the same thing as thread.sleep(). Instead of thread.sleep(), I could use other functions such as Wait() or Notify(). By using these functions, I'll save on CPU cycles that could be used for other tasks of my program.

Security Issues:

How was it present within my project?

Because my server is constantly listening for messages from a client on a specified IP Address and port, it makes it vulnerable for anyone to "drop in" and steal information as 1) the server is constantly running and 2) there's no security measures to protect the information being sent. With this vulnerability, my project could be hit with a DDOS attack at any minute. And aside from the potential DDOS attack, the server and client communicate via encoded/decoded plain text. Plain text communication isn't secure and could actually be intercepted by a third unwanted party.

How can I fix it?

By making the server listen-only, attackers won't be able to access the server's resources/services. Also, setting up security features such as firewalls can help limit unauthorized access to the server by restricting certain IP addresses/ports. And instead of using plain text, opting for a secure communication protocol like SSL. Unlike plain text, it uses encryption among other things to protect the data being sent/received between servers and clients.

Slow Connectivity:

How was it present within my project?

Neither the server nor the client handles any sort of errors or timeouts that could happen on the network. This could lead to slow connection/disconnection. One example is hanging connections. If a connection doesn't handle any errors/timeouts, it may remain open and hanging for the remainder of the time. Another example would be unhandled exceptions. If there's no way of properly handling these errors/timeouts, the server could crash or be unresponsive.

How can I fix it?

By implementing code that deals with error handling/timeout mechanisms, this could reduce the occurrence of hanging connections and unhandled exceptions. Also, I could ensure that I close the client's connection once the server sends the random number back to the client; this would help prevent slow connections to the server.

Capacity Concerns:

How was it present within my project?

Because there isn't a limit on how many connections can occur at once, this leads to a waste of computer resources and poor performance as those resources would be exerted to deal with the countless requests coming to and from the server. Also, capacity concerns are also present in the form of 'ignorant sending.' This means that regardless of whether a client's request is valid or not, the server will send a response back to the client even if a response was invalid or even malicious. Like the lack of a limit on the number of connections, 'ignorant sending' can also lead to a waste of computer resources and poor performance.

How can I fix it?

To fix this, I could implement a connection limit that'll limit how many connections could join at once. I could also implement some sort of function that could be used to validate each request that is being sent to the server to ensure that each request that's receiving the response isn't invalid or malicious.

Monitoring & Maintenance:

How was it present within my project?

There isn't any form of logging or monitoring for this project. This could make it difficult for me to debug my project if the project ever ran into any issues. There also isn't any sort of maintenance that'll keep my project up to date with the latest versions/upgrades. This could make it hard to maintain and update the code as I'd have to manually update the project with every new version.

How can I fix it?

By implementing some code that'll monitor and log the project for me, I could be used to track server performance and can also be used for diagnosis if there is ever an issue with the server. I could also implement some other code that could update the code in case of any future project upgrades or changes.

Results

Overall, I think I'm proud of it although it was simplistic in terms of design since I was able to carry out my vision onto Unity. Compared to the beginning of this project, it wasn't what I had expected since I had changed my project halfway. Of course, however, after changing my project, it was what I had expected. I believed that if I had more time, I would've most definitely had better results. If I was able to spend more time on the designs and applied my server to a more complex problem or an even-bigger project, then it definitely would have turned out better.

Summary

My three main takeaways are 1) To take advantage of those times of uncertainty to try and create something amazing from the seemingly impossible whenever problems arise. 2) TCP is important to server-client architecture. Without it, a server could run into many problems such as data corruption or unreliable delivery. And last, but not least, 3) have fun with projects; by aligning projects with your interests, you may enjoy working on it more.

Works Cited

Basu, Arya. "Class Google Drive." *Accounts.google.com*, drive.google.com/drive/u/3/folders/13IdhV3dwJQ4b1QvW2aKFgfQkISn-NUpi. Accessed 3 May 2023.

---. "Course Schedule Google Docs." *Docs.google.com*, 17 Jan. 2023, docs.google.com/document/d/1KFGkQtjE8BG4u2huke26U1EmzTTJcCAsbZeuv_3FwGY/edit. Accessed 3 May 2023.

"Challenges of Computer Network." *GeeksforGeeks*, 18 July 2020, www.geeksforgeeks.org/challenges-of-computer-network/.

Cyber, Elevate. "Building a TCP Client in C#." *Www.youtube.com*, www.youtube.com/watch?v=qtZTf1L5v0E&ab_channel=ElevateCyber. Accessed 3 May 2023.

"How Do I Generate a Random Integer in C#?" *Stack Overflow*, stackoverflow.com/questions/2706500/how-do-i-generate-a-random-integer-in-c.

Unity Technologies. "Unity - Unity." *Unity*, 2019, unity.com/.

Weiland, Tom. "Tom Weiland - YouTube Channel." *Www.youtube.com*, www.youtube.com/@tomweiland.