```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace Assets.Scripts
{
    //Class for representing an email inbox tab, serializable to persist across email session
s
    public class EmailInbox
    {
        //Message class for each individual message
        public class Message
        {
            [XmlAttribute]
            public string senderOrRecipient;
            [XmlAttribute]
            public string subject;
            [XmlAttribute]
            public string time;
            [XmlAttribute]
            public string text;

            //No argument constructor necessary for serialization
            public Message()
            {
                senderOrRecipient = "Sender";
                subject = "Subject";
                time = "Time";
                text = "Text";

            }

            public Message(string senderOrRecipient, string subject, string time, string text
)
            {
                this.senderOrRecipient = senderOrRecipient;
                this.subject = subject;
                this.time = time;
                this.text = text;
            }
        }

        //Tracks the type of the email inbox
        [XmlAttribute]
        public string type;

        //List of messages
```

```
        public List<Message> messageList;

        //Constructor
        public EmailInbox()
        {
            messageList = new List<Message>();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace Assets.Scripts
{
    //Class for representing an email inbox tab, serializable to persist across email session
s
    public class EmployeeData
    {
        //Message class for each individual message
        public class Employee
        {
            [XmlAttribute]
            public string name;
            [XmlAttribute]
            public string email;
            [XmlAttribute]
            public string role;

            //No argument constructor necessary for serialization
            public Employee()
            {
                name = "Doe, John";
                email = "doe.john@bmail.com";
                role = "CEO";
            }

            public Employee(string name, string email, string role)
            {
                this.name = name;
                this.email = email;
                this.role = role;
            }
        }

        //List of messages
```

```
        public List<Employee> employeeList;

        //Constructor
        public EmployeeData()
        {
            employeeList = new List<Employee>();
        }
    }
}

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.UI;
using UnityEngine.Events;

public class GameController : MonoBehaviour
{

    //Puzzle is complete once logged into managers computer after guessing password: 05
    public bool DisableManagersSeecurityProtectionPuzzle {get; set;}

    //Puzzle is complete after sending phishing email to employee from post it note computer
: 06
    //Dependent on puzzle 1 being completed, and employee files being found
    public bool PhishingPuzzle {get; set;}

    //Puzzle is complete once virus is uploaded to network: 07
    public bool VirusDisablesProtectionPuzzle {get; set;}

    //Puzzle is complete once ddos attack is sent from computer unlocked via phishing puzzle:
 08
    //Dependent on puzzle 06
    public bool DDoSPuzzle {get; set;}

    //Signals if all puzzles are complete and server room door can be opened
    public bool ServerRoomOpen {get; private set;}

    //Tracks whether the response to the phishing email from puzzle 6 has been sent
    public bool PhishingResponse { get; set; }

    //tracks whether the storage room door is unlocked
    public bool StorageUnlocked{ get; set; }

    //tracks whether the Manager's room door is unlocked
    public bool BossUnlocked{ get; set; }
```

```
    //Tracks whether the player has found the USB stick in the storage room, and whether they
have downloaded the virus onto it

    public static bool PlayerHasUsb{ get; set; }
    public static bool USBHasVirus{ get; set; }

    private bool alarmCountdownInitiated;

    private bool InLoseState;
    private bool InWinState;

    public Timer timer;
    public TMP_Text timeText;

    //Will be used once player triggers the countdown
    public GameObject redFilter;

    private GameObject redFilterInstance;

    // Start is called before the first frame update
    void Start()
    {
        //Locking cursor to the window
        Cursor.lockState = CursorLockMode.Confined;

        DontDestroyOnLoad(gameObject);
        DontDestroyOnLoad(timeText.transform.parent.gameObject);

        timer = new Timer(1800);
        ResetBooleans();
    }

    //Sets initial values for booleans
    public void ResetBooleans()
    {
        DisableManagersSeecurityProtectionPuzzle = false;
        PhishingPuzzle = false;
        VirusDisablesProtectionPuzzle = false;
        DDoSPuzzle = false;
        alarmCountdownInitiated = false;
        PhishingResponse = false;
        InLoseState = false;
        InWinState = false;
        StorageUnlocked = false;
        BossUnlocked = false;
        ServerRoomOpen = false;

    }

    // Update is called once per frame
```

```
    void Update()
    {
        timer.UpdateTime();
        DisplayTime(timer.TimeRemaining);
        if (timer.TimeRemaining <= 0) {
            //Lose State
            ActivateLoseState();
        }
        if (!alarmCountdownInitiated) {
            if (VirusDisablesProtectionPuzzle && DisableManagersSeecurityProtectionPuzzle) {
                // 5 minutes
                if(timer.TimeRemaining > 300){
                    timer.TimeRemaining = 300;
                }
                alarmCountdownInitiated = true;

                //Adds red filter
                redFilterInstance = Instantiate(redFilter);
            }
        } else {
            if(DDoSPuzzle){
                Destroy(redFilterInstance);
                ServerRoomOpen = true;
            }
        }

    }

    public void ActivateLoseState() {
        if (!InLoseState)
        {
            InLoseState = true;
            timeText.transform.parent.gameObject.SetActive(false);
            PuzzleSceneManager.SceneSwitch("LoseScene");
        }
    }

    public void ActivateWinState() {
        if (!InWinState)
        {
            InWinState = true;
            timer.timerIsRunning = false;
            timeText.transform.parent.gameObject.SetActive(false);
            PuzzleSceneManager.SceneSwitch("WinScene");
        }
    }

    private void DisplayTime(float timeToDisplay) {
        timeToDisplay += 1;
```

```
        float minutes = Mathf.FloorToInt(timeToDisplay / 60);
        float seconds = Mathf.FloorToInt(timeToDisplay % 60);

        timeText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
    }

    public void StartGame() {
        //Main room
        PuzzleSceneManager.SceneSwitch("GameScene");

        //Reset booleans
        ResetBooleans();

        //Starting timer
        timer.ResetTime();
        timeText.transform.parent.gameObject.SetActive(true);
        timer.timerIsRunning = true;
    }

    public void QuitGame() {
        PuzzleSceneManager.QuitGame();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class KeypadScriptBoss : MonoBehaviour
{
    [SerializeField] private string password;
    [SerializeField] private TextMeshProUGUI output;
    private string _currentString;

    void Start()
    {
        _currentString = "";
        this.UpdateText();
    }

    public void AddCharacter(string Char) {
        if(_currentString.Length < 7){
            _currentString += Char;
            this.UpdateText();
        } else {
            this.ClrString();
        }
    }
```

```
    public void ClrString(){
        _currentString = "";
        this.UpdateText();
    }

    public void SubmitString(){
        if(_currentString.Equals(password)){
            _currentString = "CORRECT";
            //Sets Unlocked Flag
            GameObject.FindGameObjectWithTag("GameController").GetComponent<GameController>()
.BossUnlocked = true;
            this.UpdateText();
        } else {
            this.ClrString();
        }
    }

    private void UpdateText(){
        output.text = _currentString;
    }

    public void ExitPad(){
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
using static UnityEngine.InputSystem.InputAction;

public class PlayerController : MonoBehaviour
{
    //Interaction data, public, to be set in Unity editor
    public InteractionInputData interaction;

    //Character controller for movement, collision
    private CharacterController cc;

    //Camera
    private GameObject cameraObject;

    //Vector variables for storing input
    private Vector2 movementVector;
    private Vector2 cameraVector;
    private bool clicked;

    //Settings for move speed and camera sensitivity, public to allow changing in Unity
    public float speed = 50f;
```

```
    public float cameraSensititivity = 10;

    //Setting for gravity
    public float gravity = -10;

    void Start()
    {
        //Get the character controller and camera
        cc = GetComponent<CharacterController>();
        cameraObject = Camera.main.gameObject;

        //Set up interaction scriptable object
        interaction.Reset();
    }

    //Event handlers for input
    private void OnMove(CallbackContext context)
    {
        movementVector = context.ReadValue<Vector2>();
    }

    public void OnLook(CallbackContext context)
    {
        cameraVector = context.ReadValue<Vector2>();
    }

    public void OnInteract(CallbackContext context)
    {
        bool data = context.ReadValueAsButton();

        //Only go on when the button press/click begins
        if (data && !clicked)
        {
            interaction.p_interactPress = true;
        }

        //Track whether button is pressed or not
        clicked = data;
    }

    void Update()
    {
        //Calculating new camera rotation
        Vector3 cameraDelta = new Vector3(-
cameraVector.y, 0, 0) * Time.deltaTime * cameraSensititivity;
        Vector3 cameraRotation = cameraObject.transform.rotation.eulerAngles;
        cameraRotation += cameraDelta;

        //Updating camera rotation
        cameraObject.transform.rotation = Quaternion.Euler(cameraRotation);
```

```
        //Calculating new body rotation
        Vector3 bodyRotationDelta = new Vector3(0, cameraVector.x, 0) * Time.deltaTime * came
raSensititivity;
        Vector3 bodyRotation = gameObject.transform.rotation.eulerAngles;
        bodyRotation += bodyRotationDelta;

        //Updating body rotation
        gameObject.transform.rotation = Quaternion.Euler(bodyRotation);

        //Applying movement
        Vector3 movementDelta = new Vector3(movementVector.x, 0, movementVector.y) * Time.del
taTime * speed;
        movementDelta += new Vector3(0, gravity, 0) * Time.deltaTime;
        movementDelta = transform.TransformDirection(movementDelta);
        cc.Move(movementDelta * 0.1f);
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using static UnityEngine.InputSystem.InputAction;

public class BossPasswordScript : MonoBehaviour
{
    public TMP_InputField password;

    //Probably should be private readonly, but we still need to figure out the final password
, so leave public to change and test in Unity for now
    public string expectedPassword;

    public void OnClick()
    {
        if (password.text == expectedPassword)
        {
            //Disable interaction
            password.interactable = false;

            //Go to email puzzle
            PuzzleSceneManager.SwitchToPuzzle("ShutdownSecurityScene");
        }
        password.text = "";
    }

    public void Exit()
    {
        PuzzleSceneManager.ExitPuzzle();
    }
```

```
}

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using static UnityEngine.InputSystem.InputAction;

public class Puzzle1Manager : MonoBehaviour
{
    public TMP_InputField email;
    public TMP_InputField password;

    public string expectedEmail;
    public string expectedPassword;

    public void OnClick()
    {
        if (email.text == expectedEmail && password.text == expectedPassword)
        {
            GameObject controller = GameObject.FindGameObjectWithTag("GameController");

            //Disable interaction
            password.interactable = false;

            //Go to email puzzle
            PuzzleSceneManager.SwitchToPuzzle("6.PhishingEmail");
        }
    }

    //Closes the puzzle
    public void OnClose()
    {
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class Puzzle2HiddenMessage : MonoBehaviour
{
    public GameObject canvas1;

    public GameObject canvas2;
```

```
    public GameObject canvas3;

    public TMP_InputField enteredCode;

    public void OnClick()
    {
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Puzzle3DecyptedScript : MonoBehaviour
{
    public void ExitPuzzle(){
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Puzzle3sslstrip : MonoBehaviour
{
    public GameObject canvas1;
    public GameObject canvas2;
    public GameObject canvas3;
    public GameObject canvas4;
    public GameObject wrongCommand1;
    public GameObject wrongCommand2;

    public void OnClickInstall()
    {
        canvas1.SetActive (false);
        canvas2.SetActive (true);
    }

    public void OnClickCanvas2()
    {
        canvas2.SetActive (false);
        canvas3.SetActive (true);
    }

    public void OnClickCanvas3()
    {
        canvas3.SetActive (false);
        canvas4.SetActive (true);
```

```
    }

    // when wrong  choices are made
    //wrong choice on canvas  2
    public void OnClickWrong1()
    {
        canvas2.SetActive (false);
        wrongCommand1.SetActive (true);
    }
    public void GoBackTo2()
    {
        wrongCommand1.SetActive (false);
        canvas2.SetActive (true);
    }
    //wrong choice on canvas  3
    public void OnClickWrong2()
    {
        canvas3.SetActive (false);
        wrongCommand2.SetActive (true);
    }
    public void GoBackTo3()
    {
        wrongCommand2.SetActive (false);
        canvas3.SetActive (true);
    }

    public void OnClickCompleteProcess() {
        PuzzleSceneManager.SwitchToPuzzle("DecryptedSSLScene");
    }
}

using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class Puzzle4PacketSniffer : MonoBehaviour
{
    public GameObject canvas1;

    public GameObject canvas2;

    public GameObject canvas3;

    public GameObject canvas4;

    public GameObject canvas5;

    public GameObject canvas6;
```

```
public TMP_InputField FirstCommand;

public TMP_InputField SecondCommand;

public TMP_InputField ThirdCommand;

public TMP_InputField FourthCommand;

public void OnEnterFirstCommand()
{
    string expectedFirstCommand = "pktmon";
    string x = FirstCommand.text;

    if (
        expectedFirstCommand
            .IndexOf(x, 0, StringComparison.CurrentCultureIgnoreCase) !=
        -1
    )
    {
        canvas1.SetActive(false);
        canvas2.SetActive(true);
    }
}

public void OnEnterSecondCommand()
{
    string expectedSecondCommand = "pktmon start --etw";
    string x = SecondCommand.text;

    if (
        expectedSecondCommand
            .IndexOf(x, 0, StringComparison.CurrentCultureIgnoreCase) !=
        -1
    )
    {
        canvas2.SetActive(false);
        canvas3.SetActive(true);
    }
}

public void OnEnterThirdCommand()
{
    string expectedThirdCommand = "pktmon stop";
    string x = ThirdCommand.text;

    if (
        expectedThirdCommand
            .IndexOf(x, 0, StringComparison.CurrentCultureIgnoreCase) !=
        -1
```

```
        )
        {
            canvas3.SetActive(false);
            canvas4.SetActive(true);
        }
    }

    public void OnEnterFourthCommand()
    {
        string expectedFourthCommand = "pktmon format pktmon.et1 -o pktmon.txt";
        string x = FourthCommand.text;

        if (
            expectedFourthCommand
                .IndexOf(x, 0, StringComparison.CurrentCultureIgnoreCase) !=
            -1
        )
        {
            canvas4.SetActive(false);
            canvas5.SetActive(true);
        }
    }

    public void OnOpenFile()
    {
        canvas5.SetActive(false);
        canvas6.SetActive(true);
    }

    public void ExitPuzzle(){
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Puzzle5PictureManager : MonoBehaviour
{
    public void LeavePicture(){
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System;
```

```
using UnityEngine.UI;
using System.Xml;
using System.Xml.Serialization;
using Assets.Scripts;
using System.IO;

public class Puzzle6Manager : MonoBehaviour
{
    //Game controller object
    private GameObject gameController;

    //Message data
    public EmailInbox inboxData;
    public EmailInbox sentData;
    public EmailInbox spamData;
    public EmailInbox trashData;

    //Send button
    public Button sendButton;

    //Input fields
    public TMP_InputField toField;
    public TMP_InputField subjectField;
    public TMP_InputField messageField;

    //Target desired data, can be set in editor
    public string toTarget;
    public string[] subjectTargets;
    public string[] messageTargets;

    //Target word match counts
    public int subjectMatchCountTarget;
    public int messageMatchCountTarget;

    //Email GameObject references
    public GameObject emailComposeWindow;
    public GameObject emailMessageList;
    public GameObject emailMessageText;

    //Message item prefab for instantiating list
    public GameObject message;

    void Start()
    {
        //Get the game controller
        gameController = GameObject.FindGameObjectWithTag("GameController");

        InitializeMessages();

    }
```

```
    //Event handler for closing email
    public void OnClickEmailClose()
    {
        emailMessageText.SetActive(false);
    }

    //Event handler for closing compose window
    public void OnClickComposeClose()
    {
        emailComposeWindow.SetActive(false);
    }

    //Event handler for closing window
    public void OnClose()
    {
        PuzzleSceneManager.ExitPuzzle();
    }

    //Event handler for opening inboxes
    public void OnClickEmailCategory(string type)
    {
        EmailInbox currentData = inboxData;

        switch (type)
        {
            case "Sent":
                currentData = sentData;
                break;
            case "Spam":
                currentData = spamData;
                break;
            case "Trash":
                currentData = trashData;
                break;
        }

        InstantiateMessages(currentData);
    }

    //Event handler for opening an email
    public void EmailItemEvent(EmailInbox sourceList, int index)
    {
        emailMessageText.SetActive(true);

        //Special case email was from sent
        if (sourceList.type == "Sent")
        {
            emailMessageText.transform.GetChild(0).GetChild(0).GetComponent<TMP_Text>().text
= "martens.alice@bmail.com";
```

```
            emailMessageText.transform.GetChild(1).GetChild(0).GetComponent<TMP_Text>().text
= sourceList.messageList[index].senderOrRecipient;

        } else
        {
            emailMessageText.transform.GetChild(0).GetChild(0).GetComponent<TMP_Text>().text
= sourceList.messageList[index].senderOrRecipient;
            emailMessageText.transform.GetChild(1).GetChild(0).GetComponent<TMP_Text>().text
= "martens.alice@bmail.com";
        }

        emailMessageText.transform.GetChild(2).GetChild(0).GetComponent<TMP_Text>().text = so
urceList.messageList[index].subject;
        emailMessageText.transform.GetChild(3).GetChild(0).GetComponent<TMP_Text>().text = so
urceList.messageList[index].text.Replace("~", Environment.NewLine);
    }

    //Populate message list with messages
    void InstantiateMessages(EmailInbox messageList)
    {
        //Clear list
        for(int i = 0; i < emailMessageList.transform.childCount; i++)
        {
            Destroy(emailMessageList.transform.GetChild(i).gameObject);
        }

        //Add messages
        for (int i = 0; i < messageList.messageList.Count; i++)
        {
            GameObject messageObj = Instantiate(message, emailMessageList.transform);
            messageObj.GetComponent<RectTransform>().anchoredPosition = new Vector3(0, -
47 - 20 * i, 0);
            messageObj.transform.GetChild(0).GetComponent<TMP_Text>().SetText(messageList.mes
sageList[i].senderOrRecipient);
            messageObj.transform.GetChild(1).GetComponent<TMP_Text>().SetText(messageList.mes
sageList[i].subject);
            messageObj.transform.GetChild(2).GetComponent<TMP_Text>().SetText(messageList.mes
sageList[i].time);

            //Must make a copy of i, else the EmailItemEvent will break
            int index = i;
            messageObj.transform.GetChild(3).GetComponent<Button>().onClick.AddListener(() =>
 EmailItemEvent(messageList, index));
        }
    }

    //Load messages from data into EmailInboxes
    void InitializeMessages()
    {
        //Inbox
```

```
        XmlSerializer serialize = new XmlSerializer(typeof(EmailInbox));
        FileStream file = new FileStream(Application.streamingAssetsPath + "/Data/inbox.xml",
 FileMode.Open);
        inboxData = (EmailInbox)serialize.Deserialize(file);

        //Sent
        file = new FileStream(Application.streamingAssetsPath + "/Data/sent.xml", FileMode.Op
en);

        //If there is new sent data, then use that instead
        if (File.Exists(Application.streamingAssetsPath + "/Data/newsent.xml"))
        {
            file = new FileStream(Application.streamingAssetsPath + "/Data/newsent.xml", File
Mode.Open);
        }
        sentData = (EmailInbox)serialize.Deserialize(file);

        //Spam
        file = new FileStream(Application.streamingAssetsPath + "/Data/spam.xml", FileMode.Op
en);
        spamData = (EmailInbox)serialize.Deserialize(file);

        //Trash
        file = new FileStream(Application.streamingAssetsPath + "/Data/trash.xml", FileMode.O
pen);
        trashData = (EmailInbox)serialize.Deserialize(file);

        //More sanity checking
        if (gameController != null)
        {
            if (gameController.GetComponent<GameController>().PhishingResponse && gameControl
ler.GetComponent<GameController>().PhishingPuzzle)
            {
                //Load phishing response
                XmlSerializer ser = new XmlSerializer(typeof(EmailInbox.Message));
                file = new FileStream(Application.streamingAssetsPath + "/Data/phishingrespon
se.xml", FileMode.Open);
                EmailInbox.Message phishingResponse = (EmailInbox.Message)ser.Deserialize(fil
e);

                //Get time for the response
                phishingResponse.time = GetTime();

                //Add to inbox
                inboxData.messageList.Insert(0, phishingResponse);
            }
        }

        //Default to inbox
        InstantiateMessages(inboxData);
```

```
    }

    //Event handler for opening compose window
    public void OnClickCompose()
    {
        emailComposeWindow.SetActive(true);
    }

    //Event handler for send
    public void OnClickSend()
    {
        //Sequentially check if any conditions failed
        bool conditionsPassed = true;

        //Check to field, must be exact
        string to = toField.text;
        if (to != toTarget)
        {
            conditionsPassed = false;
        }

        //Check subject, need to have a certain number of keywords
        string subject = subjectField.text;
        int subjectMatchCount = 0;
        foreach (string targetWord in subjectTargets)
        {
            //Ignore case
            if (subject.IndexOf(targetWord, 0, StringComparison.CurrentCultureIgnoreCase) !=
-1)
            {
                subjectMatchCount++;
            }
        }
        //Failed
        if (subjectMatchCount < subjectMatchCountTarget)
        {
            conditionsPassed = false;
        }

        //Check message, need to have a certain number of keywords
        string message = messageField.text;
        int messageMatchCount = 0;
        foreach (string targetWord in messageTargets)
        {
            //Ignore case
            if (message.IndexOf(targetWord, 0, StringComparison.CurrentCultureIgnoreCase) !=
-1)
            {
                messageMatchCount++;
            }
```

```
        }
        //Failed
        if (messageMatchCount < messageMatchCountTarget)
        {
            conditionsPassed = false;
        }

        //Good email
        if (conditionsPassed)
        {
            if(gameController != null)
            {
                gameController.GetComponent<GameController>().PhishingPuzzle = true;
            }
        }

        //Adding sent email to sent list
        EmailInbox.Message newSent = new EmailInbox.Message(to, subject, GetTime(), message);
        sentData.messageList.Insert(0, newSent);

        //Serialize
        XmlSerializer serializer = new XmlSerializer(typeof(EmailInbox));
        TextWriter writer = new StreamWriter(Application.streamingAssetsPath + "/Data/newsent
.xml");
        serializer.Serialize(writer, sentData);

        //Close and clear email compose window
        toField.text = "";
        subjectField.text = "";
        messageField.text = "";
        OnClickComposeClose();
    }

    //Gets the "current" time
    string GetTime()
    {
        return "7:" + (29 + (int)(30 - gameController.GetComponent<GameController>().timer.Ti
meRemaining / 60)) + "pm";
    }

    void Update()
    {
        //Enable button if all fields have text
        if (toField.text != "" && subjectField.text != "" && messageField.text != "")
        {
            sendButton.interactable = true;
        } else
        {
            sendButton.interactable = false;
        }
```

```
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Puzzle8DDoSScript : MonoBehaviour
{
    // Start is called before the first frame update

    System.Random random;
    List<string> invalidIPs;
    List<string> validIPs;
    List<Text> IPtexts;

    public Text IP0;
    public Text IP1;
    public Text IP2;

    public Text answerStatus;
    public Text zombieCountText;

    public GameObject minigameScreen;
    public GameObject winScreen;

    int zombiesCt;
    int correctIP;
    bool needNewSet;
    float timeForSet;
    string lastAnswer;
    private static int MAX_TIME = 10;
    private static string INCORRECT = "incorrect";
    private static string CORRECT = "correct";

    void Start()
    {
        initializeLists();
        timeForSet = MAX_TIME;
        zombiesCt  = 0;
        IPtexts.Add(IP0);
        IPtexts.Add(IP1);
        IPtexts.Add(IP2);
        random = new System.Random();
        needNewSet = true;

    }

    // Update is called once per frame
```

```
    void Update()
    {
        if(timeForSet > 0) {
            timeForSet -= Time.deltaTime;

        }else {
            needNewSet = true;
        }
        if (needNewSet) {
            setIPs();
            needNewSet = false;
            timeForSet = MAX_TIME;
            zombieCountText.text = "zombies - " + zombiesCt;
            answerStatus.text = lastAnswer;
        }

        if (zombiesCt > 9) {
            minigameScreen.SetActive(false);
            winScreen.SetActive(true);

            //Set DDoS puzzle complete
            GameObject.FindGameObjectWithTag("GameController").GetComponent<GameController>()
.DDoSPuzzle = true;
            StartCoroutine(LeaveScene());

        }
    }

    private IEnumerator LeaveScene()
    {
        yield return new WaitForSeconds(10);
        GameObject.FindGameObjectWithTag("GameController").GetComponent<GameController>().DDo
SPuzzle = true;
        PuzzleSceneManager.ExitPuzzle();
    }

    private void setIPs() {
        correctIP = random.Next(3);
        for(int i=0; i < 3; i++) {
            if (i == correctIP) {
                IPtexts[i].text = validIPs[random.Next(validIPs.Count)];
            }
            else {
                IPtexts[i].text = invalidIPs[random.Next(invalidIPs.Count)];
            }
        }
    }

    public void OnClickComputer0() {
        if (correctIP == 0) {
```

```
            zombiesCt++;
            lastAnswer = CORRECT;
        }
        else {
            lastAnswer = INCORRECT;
        }
        needNewSet = true;
    }

    public void OnClickComputer1() {
        if (correctIP == 1) {
            zombiesCt++;
            lastAnswer = CORRECT;
        }
        else {
            lastAnswer = INCORRECT;
        }
        needNewSet = true;
    }

    public void OnClickComputer2() {
        if (correctIP == 2) {
            zombiesCt++;
            lastAnswer = CORRECT;
        }
        else {
            lastAnswer = INCORRECT;
        }
        needNewSet = true;
    }

    private void initializeLists() {
        invalidIPs = new List<string>();
        validIPs = new List<string>();
        IPtexts = new List<Text>();

        //load incorrect addresses
        invalidIPs.Add("0.1.2.3.4");
        invalidIPs.Add("12.38.2.1000");
        invalidIPs.Add("12.0..540");
        invalidIPs.Add("12.2.330.9");
        invalidIPs.Add("12.10.10.10.");
        invalidIPs.Add("12.5.90.900");
        invalidIPs.Add("1234.0.0.0");
        invalidIPs.Add("12..255.255.255");
        invalidIPs.Add("12.122.176");
        invalidIPs.Add("12.01.250.12");
        invalidIPs.Add("12.10.010.100");
        invalidIPs.Add("12.9.74.1.14");
        invalidIPs.Add("12.9.256.128");
```

```
        invalidIPs.Add("12.128.128");
        invalidIPs.Add("12.1y.15.0");
        invalidIPs.Add("12.E.15.120");
        invalidIPs.Add("12.67.15.t4");
        invalidIPs.Add("12.14.1024.1");
        invalidIPs.Add("12.36.2011");
        invalidIPs.Add("12.99.9f.11");

        //load correct addresses
        validIPs.Add("12.12.12.12");
        validIPs.Add("12.255.255.12");
        validIPs.Add("12.10.10.255");
        validIPs.Add("1.2.3.4");
        validIPs.Add("12.100.0.3");
        validIPs.Add("144.11.7.7");
        validIPs.Add("1.1.1.1");
        validIPs.Add("12.248.15.79");
        validIPs.Add("12.0.0.12");
        validIPs.Add("12.99.103.3");
        validIPs.Add("12.33.66.99");
        validIPs.Add("12.120.12.120");
        validIPs.Add("0.0.0.12");
        validIPs.Add("12.19.20.21");
        validIPs.Add("2.2.2.4");
        validIPs.Add("8.88.255.88");
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Puzzle8UnlockComputer : MonoBehaviour
{
    public TMP_InputField username;
    public TMP_InputField password;

    public string expectedUsername;
    public string expectedPassword;

    public void OnClick()
    {
        Debug.Log("clicked");
        if (username.text == expectedUsername && password.text == expectedPassword)
        {
            //Disable interaction
            password.interactable = false;

            //Go to email puzzle
```

```
            PuzzleSceneManager.SwitchToPuzzle("8.DDoSScene");
        }
        password.text = "";
    }

    public void Exit()
    {
        PuzzleSceneManager.ExitPuzzle();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

//Static class for handling switching between scenes
public class PuzzleSceneManager
{
    //Track whether in a puzzle or not
    private static bool inPuzzle = false;

    //Name of puzzle scene in use
    private static string sceneString;

    private static GameObject camera;
    private static GameObject player;

    //Switching non-puzzle scenes
    public static void SceneSwitch(string scene)
    {
        SceneManager.LoadSceneAsync(scene, LoadSceneMode.Single);

        sceneString = scene;
    }

    //Switches to the puzzle scene specified
    public static void SwitchToPuzzle(string scene)
    {
        //Entering puzzle from main scene
        if (!inPuzzle)
        {
            //Additive to keep the previous scene loaded
            AsyncOperation operation = SceneManager.LoadSceneAsync(scene, LoadSceneMode.Addit
ive);
            operation.allowSceneActivation = true;

            sceneString = scene;
            inPuzzle = true;
```

```
            PauseScene();
        } else //Entering puzzle from a puzzle
        {
            //Additive to keep the previous scene loaded
            AsyncOperation operation = SceneManager.LoadSceneAsync(scene, LoadSceneMode.Addit
ive);

            //Only unload after the new scene completes loading, need to pass new scene strin
g
            operation.completed += sender => PuzzleSwitch(scene);
            operation.allowSceneActivation = true;

            inPuzzle = true;
        }
    }

    //Exits the puzzle, resumes the previous scene
    public static void ExitPuzzle()
    {
        //Sanity check that we are not trying to exit a puzzle when there is no puzzle
        if (inPuzzle)
        {
            AsyncOperation operation = SceneManager.UnloadSceneAsync(sceneString);
            operation.completed += UnloadDone;

            inPuzzle = false;
        }
    }

    //Event handler for when scene is unloaded
    private static void UnloadDone(AsyncOperation operation)
    {
        //Only unpause when game is done unloading the puzzle
        UnPauseScene();
    }

    //Event handler for switching puzzles
    private static void PuzzleSwitch(string scene)
    {
        //Only unload previous puzzle once game loads the new one
        SceneManager.UnloadSceneAsync(sceneString);

        //Only set new scene string after the old one is used for unloaded
        sceneString = scene;
    }

    //Pauses current scene
    private static void PauseScene()
    {
        //Should run only once when pause is first used
```

```
        if (player == null)
        {
            player = GameObject.FindGameObjectWithTag("Player");
        }

        //Player should be set to something, just a sanity check
        if (player != null)
        {
            //Camera is a child of the player, will be disabled and enabled as a side effect
of the player
            player.SetActive(false);
        }

    }

    //Unpauses current scene
    private static void UnPauseScene()
    {
        //Should never run, should have already been set by pause, but here for sanity checki
ng
        if (player == null)
        {
            player = GameObject.FindGameObjectWithTag("Player");
        }

        //Player should be set to something, just a sanity check
        if (player != null)
        {
            //Camera is a child of the player, will be disabled and enabled as a side effect
of the player
            player.SetActive(true);
        }
    }

    public static void QuitGame() {
        Application.Quit();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using static UnityEngine.InputSystem.InputAction;

public class SecurityShutdownScript : MonoBehaviour
{
    public void OnClickPower()
    {
```

```
        GameObject.FindGameObjectWithTag("GameController").GetComponent<GameController>().Dis
ableManagersSeecurityProtectionPuzzle = true;
        PuzzleSceneManager.ExitPuzzle();
    }

    public void OnClickCancel()
    {
        PuzzleSceneManager.SwitchToPuzzle("PasswordScene");
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class TestManager : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Keyboard.current.pKey.wasPressedThisFrame)
        {
            PuzzleSceneManager.SwitchToPuzzle("1.EmailLogin");
        }
        if (Keyboard.current.oKey.wasPressedThisFrame)
        {
            PuzzleSceneManager.SwitchToPuzzle("TestPuzzle2");
        }
        if (Keyboard.current.eKey.wasPressedThisFrame)
        {
            PuzzleSceneManager.ExitPuzzle();
        }
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

// static timer class
public class Timer
{
```

```
    public float StartTime { get; set; }
    public float TimeRemaining {get; set;}
    public bool timerIsRunning = false;


    public Timer(int startTime) {
        // Starts the timer automatically
        timerIsRunning = false;
        StartTime = startTime;
        //30 minutes
        TimeRemaining = startTime;
    }

    public void ResetTime()
    {
        timerIsRunning = false;
        TimeRemaining = StartTime;
    }
    public void UpdateTime()
    {
        if (timerIsRunning)
        {
            if (TimeRemaining > 0)
            {
                TimeRemaining -= Time.deltaTime;
            }
            else
            {
                Debug.Log("Time has run out!");
                TimeRemaining = 0;
                timerIsRunning = false;
            }
        }
    }

}
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

// attach to UI Text component (with the full text already there)

public class TypeScript : MonoBehaviour
{

    Text txt;
    string story;

    void Start ()
    {
```

```
        txt = GetComponent<Text> ();
        story = txt.text;
        txt.text = "";

        StartCoroutine ("PlayText");
    }

    IEnumerator PlayText()
    {
        foreach (char c in story)
        {
            txt.text += c;
            yield return new WaitForSeconds (0.075f);
        }
    }

}


using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
public class WinController : MonoBehaviour
{
    public GameControllerProxy proxy;
    public TMP_Text timeText;
    // Start is called before the first frame update
    void Start()
    {
        Timer timer = proxy.GetTimer();
        float elapsedTime = timer.StartTime - timer.TimeRemaining;
        float minutes = Mathf.FloorToInt(elapsedTime / 60);
        float seconds = Mathf.FloorToInt(elapsedTime % 60);

        timeText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DoorInteractionsBoss : MonoBehaviour, IInteractable
{
    [Header("Data Objects")]
    [SerializeField] private string lockedHoverMessage;
    [SerializeField] private string unlockedHoverMessage;
    [SerializeField] private string scene;
```

```
    //interavtability variable, set based on interaction
    private bool isInteractable;
    private Animator _anim;
    private GameObject gameController;

    public bool IsInteractable { get => isInteractable; }

    public string HoverMessage {
        get {
            if(!gameController.GetComponent<GameController>().BossUnlocked){
                return lockedHoverMessage;
            } else {
                return unlockedHoverMessage;
            }
        }
    }

    public void onInteract()
    {
        //Debug.Log("" + gameController.GetComponent<GameController>().BossUnlocked);
        if(!gameController.GetComponent<GameController>().BossUnlocked){
            PuzzleSceneManager.SwitchToPuzzle(scene);
        } else {
            _anim.SetBool("hasBeenOpened", true);
            isInteractable = false;
        }
    }

    void Start()
    {
        isInteractable = true;
        _anim = this.transform.GetComponent<Animator>();
        gameController = GameObject.FindGameObjectWithTag("GameController");
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DoorInteractionsStorage : MonoBehaviour, IInteractable
{
    [Header("Data Objects")]
    [SerializeField] private string lockedHoverMessage;
    [SerializeField] private string unlockedHoverMessage;
    [SerializeField] private string scene;

    //interavtability variable, set based on interaction
    private bool isInteractable;
    private Animator _anim;
```

```
    private GameObject gameController;

    public bool IsInteractable { get => isInteractable; }

    public string HoverMessage {
        get {
            if(!gameController.GetComponent<GameController>().StorageUnlocked){
                return lockedHoverMessage;
            } else {
                return unlockedHoverMessage;
            }
        }
    }

    public void onInteract()
    {
        if(!gameController.GetComponent<GameController>().StorageUnlocked){
            PuzzleSceneManager.SwitchToPuzzle(scene);
        } else {
            _anim.SetBool("hasBeenOpened", true);
            isInteractable = false;
        }
    }

    void Start()
    {
        isInteractable = true;
        _anim = this.transform.GetComponent<Animator>();
        gameController = GameObject.FindGameObjectWithTag("GameController");
    }

}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HackerComputerIneractable : MonoBehaviour, IInteractable
{
    [Header("Data Objects")]
    private bool _hasHadUSB;
    [SerializeField] private string hoverMessageBeforeUSB;
    [SerializeField] private string hoverMessageAfterUSB;

    public bool IsInteractable {
        get{
            if(!_hasHadUSB){
                return true;
            } else {
                return true;
```

```
            }
        }
    }

    public string HoverMessage {
        get{
            if(!_hasHadUSB){
                return hoverMessageBeforeUSB;
            } else {
                return hoverMessageAfterUSB;
            }
        }
    }

    public void onInteract()
    {
        if(!_hasHadUSB && GameController.PlayerHasUsb){
            _hasHadUSB = true;
            GameController.USBHasVirus = true;
        } else {
            //Intentionally Empty
        }
    }

    void Start()
    {
        _hasHadUSB = false;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//Edited version of code from tutorial by VeryHotShark on youtube

public interface IInteractable
{
    bool IsInteractable{ get; }
    string HoverMessage{ get; }
    void onInteract();
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OpenPuzzleInteraction : MonoBehaviour, IInteractable
{
    [Header("Data Objects")]
    [SerializeField] private bool isInteractable;
```

```
    [SerializeField] private string hoverMessage;
    [SerializeField] private string scene;

    public bool IsInteractable { get => isInteractable; }

    public string HoverMessage { get => hoverMessage; }

    public void onInteract()
    {
        PuzzleSceneManager.SwitchToPuzzle(scene);
    }

    void Start()
    {
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class USBInteractable : MonoBehaviour, IInteractable
{
    [Header("Data Objects")]
    private bool isInteractable;
    [SerializeField] private string hoverMessage;

    public bool IsInteractable { get => isInteractable; }

    public string HoverMessage { get => hoverMessage; }

    public void onInteract()
    {
        isInteractable = false;
        GameController.PlayerHasUsb = true;

        gameObject.SetActive(false);
    }

    void Start()
    {
        isInteractable = true;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class VirusComputerScript : MonoBehaviour, IInteractable
{
```

```
    [Header("Data Objects")]
    [SerializeField] private bool isInteractable;
    [SerializeField] private string hoverMessage;
    [SerializeField] private string hoverMessageConfirm;

    //flag for if this is the first time, to allow for the player to confirm before final cou
ntdown starts
    private bool _hasClickedBefore;

    private GameObject gameController;

    public bool IsInteractable { get => isInteractable; }

    public string HoverMessage {
        get{
            if(!_hasClickedBefore){
                return hoverMessage;
            } else {
                return hoverMessageConfirm;
            }
        }
    }

    public void onInteract()
    {
        //player has usb with virus
        if(GameController.PlayerHasUsb && GameController.USBHasVirus){
            if(_hasClickedBefore)
            {
                //remove one of the locks
                gameController.GetComponent<GameController>().VirusDisablesProtectionPuzzle =
 true;
                gameController.GetComponent<GameController>().PhishingResponse = true;
                isInteractable = false;
            } else {
                _hasClickedBefore = true;
            }
        }
    }

    void Start()
    {
        gameController = GameObject.FindGameObjectWithTag("GameController");
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class WinInteractable : MonoBehaviour, IInteractable
{
    [Header("Data Objects")]
    [SerializeField] private bool isInteractable;
    [SerializeField] private string hoverMessage;

    private GameObject gameController;

    public bool IsInteractable { get => isInteractable; }

    public string HoverMessage { get => hoverMessage; }

    public void onInteract()
    {
        Debug.Log("ServerRoom Flag:" + gameController.GetComponent<GameController>().ServerRo
omOpen);
        //Debug.Log("Alarm Flag:" + gameController.GetComponent<GameController>().ServerRoomO
pen);
        Debug.Log("DDOS flag:" + gameController.GetComponent<GameController>().DDoSPuzzle);
        //if server room is open
        if(gameController.GetComponent<GameController>().ServerRoomOpen){
            //win the game
            gameController.GetComponent<GameController>().ActivateWinState();
        }
    }

    void Start()
    {
        gameController = GameObject.FindGameObjectWithTag("GameController");
    }
}
```