PUCMM

Pontificia Universidad Católica
Madre y Maestra

Pontificia Universidad Católica y Maestra

Campus Santo Domingo

Periodo 1-2021-2022

Willis Polanco

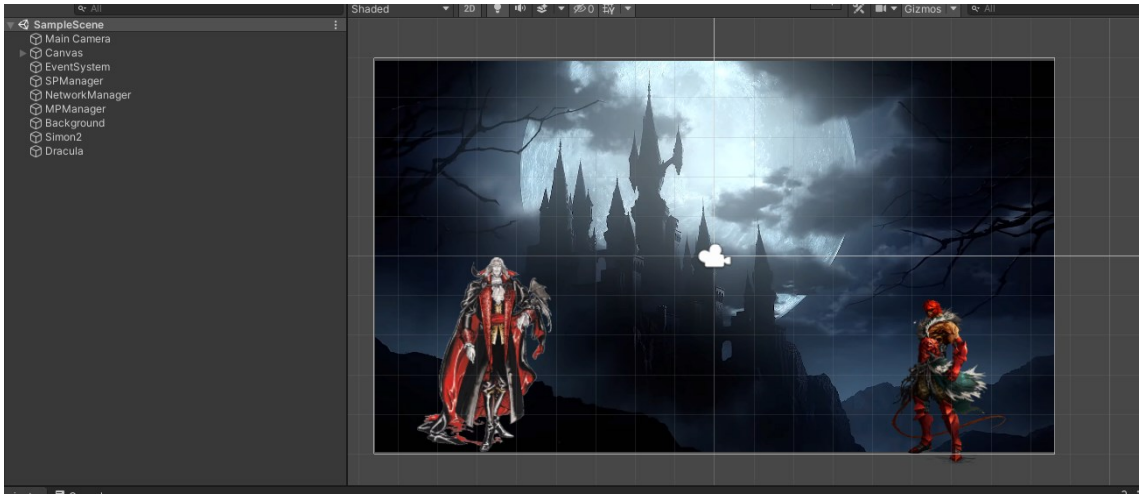
Programacion Aplicada

Práctica 2

Luis Enrique Rodríguez Varona 2018-6309

2/11/2021

Para este proyecto decidí implementar el tic tac toe en unity y utilizar C# como lenguaje.



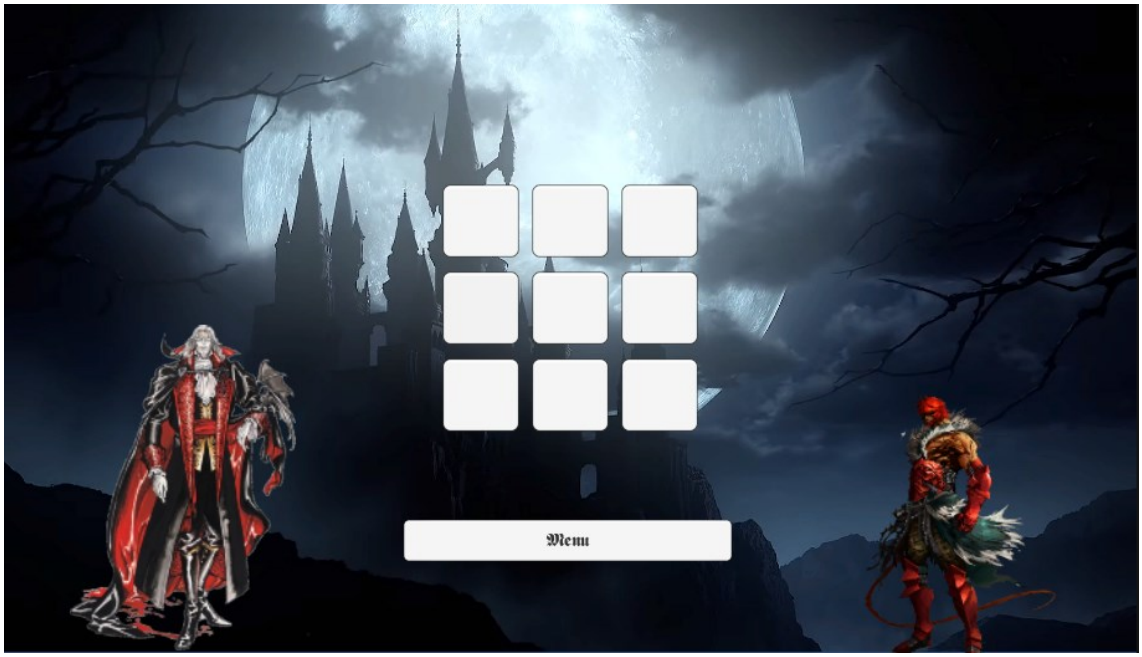
Aquí se puede ver la Escena Principal* y única del donde ocurrirá todo el juego.

Como se nos pedía desarrollar un videojuego de tictactoe, que además tuviera la función de multijugador y que también se pudiera jugar en modo single player contra la maquina se crearon los respectivos scripts para poder empezar a implementar dichas funciones. Como todo proceso en unity es muy importante declarar cada uno de los objetos con los que se va a trabajar, una vez realizado esto simplemente he de asignarles los scripts a cada uno de los objetos para que puedan tener una función determinada en la escena.

Ahora voy a explicar el script main. Básicamente este es el encargado de controlar el tema de los botones o sea los mensajes, que unos botones se apaguen que otros se enciendan, y qué hacen los botones relacionados con el menú principal, el menú de multijugador, el menú de un de un solo jugador etcétera. Aquí también se le asignó las variables para reiniciar cada uno de los otros botones puesto que en este código en vez de implementar una matriz de 3 por 3 para trabajar el TIC TAC TOE decidí implementar cada botón de lo que vendría a representar cada cuadrante de una matriz.

```
SPManager_Instance.GetComponent<SPManager>().randomTurn = Random.Range(0, 2);

if (SPManager_Instance.GetComponent<SPManager>().randomTurn == 0)
    SPManager_Instance.GetComponent<SPManager>().PlayerTurn = true;
else
    SPManager_Instance.GetComponent<SPManager>().PlayerTurn = false;

for (int i = 0; i < 9; i++)
{
    SPManager_Instance.GetComponent<SPManager>().SPTexts[i].text = "";
}

SPManager_Instance.GetComponent<SPManager>().pressed1 = false;
SPManager_Instance.GetComponent<SPManager>().pressed2 = false;
SPManager_Instance.GetComponent<SPManager>().pressed3 = false;
SPManager_Instance.GetComponent<SPManager>().pressed4 = false;
SPManager_Instance.GetComponent<SPManager>().pressed5 = false;
SPManager_Instance.GetComponent<SPManager>().pressed6 = false;
SPManager_Instance.GetComponent<SPManager>().pressed7 = false;
SPManager_Instance.GetComponent<SPManager>().pressed8 = false;
SPManager_Instance.GetComponent<SPManager>().pressed9 = false;
SPManager_Instance.GetComponent<SPManager>().End = false;

SPManager_Instance.GetComponent<SPManager>().Message.GetComponent<Text>().text = "";

// ---------CAMBIOS AL MULTIPLAYER--------

for (int i = 0; i < 9; i++)
{
    MPManager_Instance.GetComponent<MPManager>().MPTexts[i].text = "";
}

MPManager_Instance.GetComponent<MPManager>().pressed1 = false;
MPManager_Instance.GetComponent<MPManager>().pressed2 = false;
MPManager_Instance.GetComponent<MPManager>().pressed3 = false;
MPManager_Instance.GetComponent<MPManager>().pressed4 = false;
MPManager_Instance.GetComponent<MPManager>().pressed5 = false;
MPManager_Instance.GetComponent<MPManager>().pressed6 = false;
MPManager_Instance.GetComponent<MPManager>().pressed7 = false;
MPManager_Instance.GetComponent<MPManager>().pressed8 = false;
MPManager_Instance.GetComponent<MPManager>().pressed9 = false;
MPManager_Instance.GetComponent<MPManager>().End = false;

MPManager_Instance.GetComponent<MPManager>().Message.GetComponent<Text>().text = "";

Network_Instance.GetComponent<NetworkManagerHUD>().showGUI = false;
```

```
public void SinglePlayerWindow()
{
    Header.SetActive(false);
    MainMenu.SetActive(false);
    MenuButton.SetActive(true);

    VsDracula.SetActive(true);

    SinglePlayer = true;
}



public void HardMode()
{

    VsDracula.SetActive(false);
    GameButtons.SetActive(true);

    SPManager_Instance.GetComponent<SPManager>().vsDracula = true;
}

void SPGameEnded()
{
    if (SPManager_Instance.GetComponent<SPManager>().End)
    {
        RestartButton.SetActive(true);
        MenuButton.SetActive(true);
    }
}

public void MultiPlayerWindow()
{
    Header.SetActive(false);
    MenuButton.SetActive(true);
    MainMenu.SetActive(false);

    SinglePlayer = false;

    Network_Instance.GetComponent<NetworkManagerHUD>().showGUI = true;
}
```



Ahora hablar del apartado de del single Player mode, para este apartado se creó una clase que se llamaría SPManager en esta clase se declararían las acciones del jugador contra la máquina y también la inteligencia artificial de la misma. Para realizar la inteligencia artificial de la máquina aquí se declararon unos arreglos y una variable Random que dependiendo de las posiciones que estarían de introducidas en esos arreglos la inteligencia artificial decidiría dónde empezar y qué jugadas realizar dependiendo donde se empezara.

los 3 arreglos previamente mencionados serían cuando se empieza en una esquina, cuando no se empiezan en una esquina y cuando la máquina es la primera en jugar. Luego se inicializa la inteligencia en la función start donde puede depender si sí o no es la máquina la que empezó a jugar. Cree una función que se llama GameSens que es la que se encarga de determinar cuando el jugador jugó en un botón o sea en una posición determinada y en base a esto y la máquina reconoce esta posición y puede hacer una contra jugada previamente impuesta en su código.

No voy a indagar mucho sobre la cantidad de jugadas posibles que puede realizar la máquina puesto que realmente cree una gran cantidad considerable, ya que utilicé un recurso de una página web que determinan todas las jugadas posibles y lo único que tuve que hacer después era simplemente tener ya el script principal escrito y lo único que era hacer el copy paste cambiándole los valores con las posibles jugadas y contra jugadas. A esta inteligencia artificial le decidí llamar VsDrácula porque la temática del videojuego es un TIC TAC TOE en honor a Castlevania.

```
int random;
public int randomTurn;
int[] corners = { 0, 2, 6, 8 };
int[] notcorners = { 1, 3, 5, 7 };
int[] firstPlay = { 0, 2, 4, 6, 8 };

void Start()
{
    randomTurn = Random.Range(0, 2);

    if (randomTurn == 0)
        PlayerTurn = true;
    else
        PlayerTurn = false;
}

void Update()
{
    if (Main_Instance.GetComponent<Main>().SinglePlayer)
    {
        if (!PlayerWon() && !MachineWon() && !End)
            GameSense();
        else
            MatchMessage();
    }


    SPButtons[0].GetComponent<Button>().onClick.AddListener(Play1);
    SPButtons[1].GetComponent<Button>().onClick.AddListener(Play2);
    SPButtons[2].GetComponent<Button>().onClick.AddListener(Play3);
    SPButtons[3].GetComponent<Button>().onClick.AddListener(Play4);
    SPButtons[4].GetComponent<Button>().onClick.AddListener(Play5);
    SPButtons[5].GetComponent<Button>().onClick.AddListener(Play6);
    SPButtons[6].GetComponent<Button>().onClick.AddListener(Play7);
    SPButtons[7].GetComponent<Button>().onClick.AddListener(Play8);
    SPButtons[8].GetComponent<Button>().onClick.AddListener(Play9);

}
```

```
void VsDracula()
{
    for(int i = 0; i<100; i++)
    {
        random = Random.Range(0, 9);
        //primera jugada (si empieza la maquina)
        if (SPTexts[0].text == "" && SPTexts[1].text == "" && SPTexts[2].text == "" && SPTexts[3].text == "" && SPTexts[4].text == ""
            && SPTexts[5].text == "" && SPTexts[6].text == "" && SPTexts[7].text == "" && SPTexts[8].text == "")
        {
            random = Random.Range(0, 5);
            SPTexts[firstPlay[random]].text = "O";
            break;
        }
        //jugada en el centro
        else if(SPTexts[4].text == "")
        {
            SPTexts[4].text = "O";
            break;
        }
        //jugada en las esquinas
        else if (SPTexts[4].text == "X" && SPTexts[0].text == "" && SPTexts[2].text == "" && SPTexts[6].text == "" && SPTexts[8].text == "")
        {
            random = Random.Range(0, 4);
            SPTexts[corners[random]].text = "O";
            break;
        }
        //evitar perder cuando juegan en el medio
        else if(SPTexts[4].text == "X" && SPTexts[0].text == "O" && SPTexts[8].text == "X" && SPTexts[1].text == "" && SPTexts[2].text == ""
            && SPTexts[3].text == "" && SPTexts[5].text == "" && SPTexts[6].text == "" && SPTexts[7].text == "")
        {
            SPTexts[6].text = "O";
            break;
        }
        else if (SPTexts[4].text == "X" && SPTexts[2].text == "O" && SPTexts[6].text == "X" && SPTexts[0].text == "" && SPTexts[1].text == ""
            && SPTexts[3].text == "" && SPTexts[5].text == "" && SPTexts[7].text == "" && SPTexts[8].text == "")
        {
            SPTexts[8].text = "O";
            break;
        }
        else if (SPTexts[4].text == "X" && SPTexts[8].text == "O" && SPTexts[0].text == "X" && SPTexts[1].text == "" && SPTexts[2].text == ""
            && SPTexts[3].text == "" && SPTexts[5].text == "" && SPTexts[6].text == "" && SPTexts[7].text == "")
        {
            SPTexts[2].text = "O";
            break;
```

Luego Cree una clase que se llama MPManager Que está básicamente los lo que se encargaba era de verificar cuando uno de los jugadores ganado perdida y lanzar los mensajes típicos de ganaste perdices etcétera.

```
bool YouWon()
{
    if ((MPTexts[0].text == "X" && MPTexts[1].text == "X" && MPTexts[2].text == "X") ||
   (MPTexts[0].text == "X" && MPTexts[4].text == "X" && MPTexts[8].text == "X") ||
   (MPTexts[0].text == "X" && MPTexts[3].text == "X" && MPTexts[6].text == "X") ||
   (MPTexts[1].text == "X" && MPTexts[4].text == "X" && MPTexts[7].text == "X") ||
   (MPTexts[2].text == "X" && MPTexts[5].text == "X" && MPTexts[8].text == "X") ||
   (MPTexts[3].text == "X" && MPTexts[4].text == "X" && MPTexts[5].text == "X") ||
   (MPTexts[6].text == "X" && MPTexts[7].text == "X" && MPTexts[8].text == "X") ||
   (MPTexts[2].text == "X" && MPTexts[4].text == "X" && MPTexts[6].text == "X"))
    {
        return true;
    }
    else
        return false;
}

bool YouLost()
{
    if ((MPTexts[0].text == "O" && MPTexts[1].text == "O" && MPTexts[2].text == "O") ||
   (MPTexts[0].text == "O" && MPTexts[4].text == "O" && MPTexts[8].text == "O") ||
   (MPTexts[0].text == "O" && MPTexts[3].text == "O" && MPTexts[6].text == "O") ||
   (MPTexts[1].text == "O" && MPTexts[4].text == "O" && MPTexts[7].text == "O") ||
   (MPTexts[2].text == "O" && MPTexts[5].text == "O" && MPTexts[8].text == "O") ||
   (MPTexts[3].text == "O" && MPTexts[4].text == "O" && MPTexts[5].text == "O") ||
   (MPTexts[6].text == "O" && MPTexts[7].text == "O" && MPTexts[8].text == "O") ||
   (MPTexts[2].text == "O" && MPTexts[4].text == "O" && MPTexts[6].text == "O"))
    {
        return true;
    }
    else
        return false;
}
```

Casi terminando pero no menos importante está la clase que se llama PlayerScript en esta

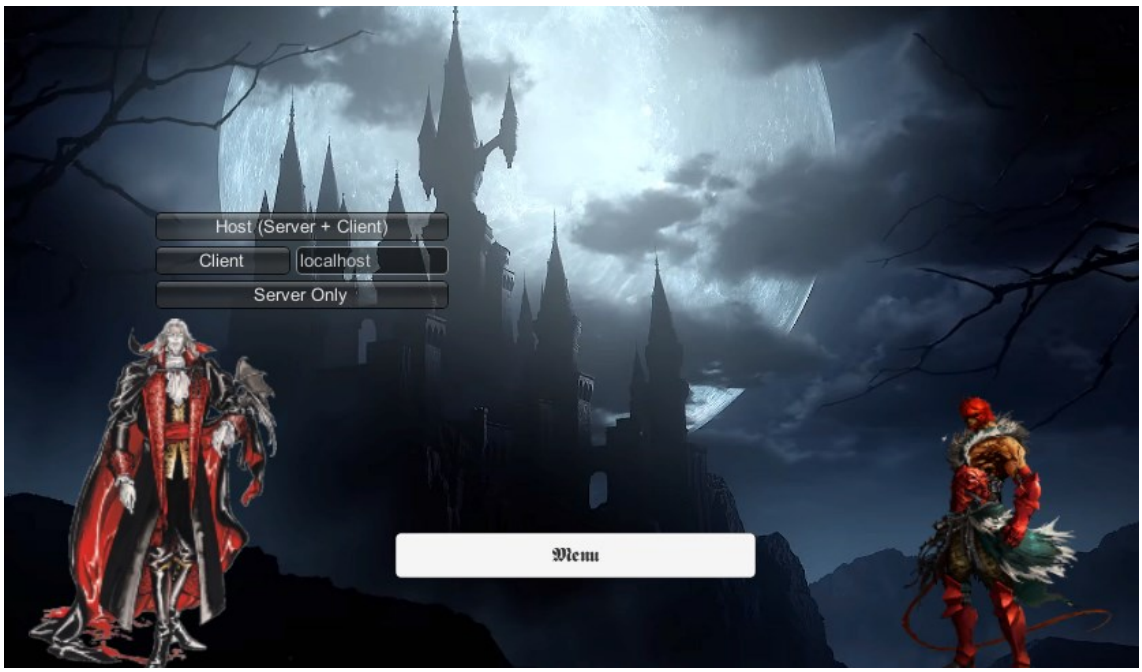
se pueden ver de una manera más detallada las relaciones de conexión en base a la librería de conexión RPC/TCP que utilice en este código para poder demostrar que hay una comunicación entre el cliente y el servidor. Aquí básicamente se demuestra cada vez que se envía una señal desde el jugador, la señal del botón el cual se jugó, la señal cuando la recibe el host etcétera. Además se crea una para cada uno de los botones que tiene la composición y ya está es simplemente eso.

```
[Command]
void IWantToPlay()
{
    if(MPManager_Instance.GetComponent<MPManager>().PlayerCount == 2)
        DisplayGamePanel();
}

[TargetRpc]
void DisplayGamePanel()
{
    MPManager_Instance.GetComponent<MPManager>().PlayerCount = 2;
    Main_Instance.GetComponent<Main>().GameButtons.SetActive(true);
    NetworkManager_Instance.GetComponent<NetworkManagerHUD>().showGUI = false;
    PanelDisplayed = true;
}

[Command]
void IncreasePlayerCount()
{
    MPManager_Instance.GetComponent<MPManager>().PlayerCount++;
}
```

Ahora sí por último tocaría hablar de cómo se establecieron las conexiones. Aquí yo lo que hice fue utilizar una librería gratuita que tiene Unity que se llama Mirror Networking. Esta librería/método tiene toda la información y básicamente es la encargada de llevar todo el tipo de comunicación entre el determinado host y el determinado de cliente. Todo se puede observar mejor tanto en la clase PlayerScript que toma esas referencias y en la otra clase que creé que se llama NetworkManagerScript, esta es una clase típica de este método que utiliza la librería de Mirror y hereda de una clase mayor que se llama Network manager que es donde están todas las referencias a todos los tipos de conexiones y demás. En esta básicamente lo que se hace crear un objeto que sería la Main_Instance quienes se tendría funciones de iniciar el servidor parar el servidor, que el cliente se conecte o que el cliente se conecte e implícitamente ya llevaría la comunicación entre estos. Lo más interesante de esto es que lo único que se tiene que hacer después, es agregarle en como componentes a los objetos en unity los componentes de Mirror, y una vez conectados se hacen las supuestas declaraciones y ya simplemente se lleva de manera casi automática las conexiones. Cabe recalcar que es una librería que se utiliza principalmente también para desarrollo de juegos online masivos o sea MMORPGs

Ahora adjuntare también imágenes del apartado de conexión del juegos y partes de lo que vendría siendo el NetworkManagerScript.

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

Script de Unity (1 referencia de recurso) | 0 referencias
public class NetworkManagerScript : NetworkManager
{
    public GameObject Main_Instance;

    5 referencias
    public override void OnStartServer()
    {
        Debug.Log("Server has started.");
        Main_Instance.GetComponent<Main>().MenuButton.SetActive(false);
    }

    2 referencias
    public override void OnStopServer()
    {
        Debug.Log("Server has stopped");
        Main_Instance.SetActive(true);
        Main_Instance.GetComponent<Main>().MultiPlayerWindow();
    }

    7 referencias
    public override void OnClientConnect(NetworkConnection conn)
    {
        Debug.Log("Client has connected");
        Main_Instance.GetComponent<Main>().MenuButton.SetActive(false);
    }

    4 referencias
    public override void OnClientDisconnect(NetworkConnection conn)
    {
        Debug.Log("Client has disconnected");
    }
}
```




Server: active. Transport: KCP
Client: address=localhost
Client Ready
Stop Host