**UNIVERSIDAD TECNOLOGICA DE SANTIAGO, (UTESA)**

**SISTEMA CORPORATIVO**

**Carrera de Ingeniería en Sistema y Computación**

# PROGRAMACION DE VIDEO JUEGOS

## TRABAJO A REALIZAR

Capítulo 3.

## DOCENTE

Ing. Iván Mendoza

## PRESENTADO POR:

Alexander Ventura Mercado 2-17-0038

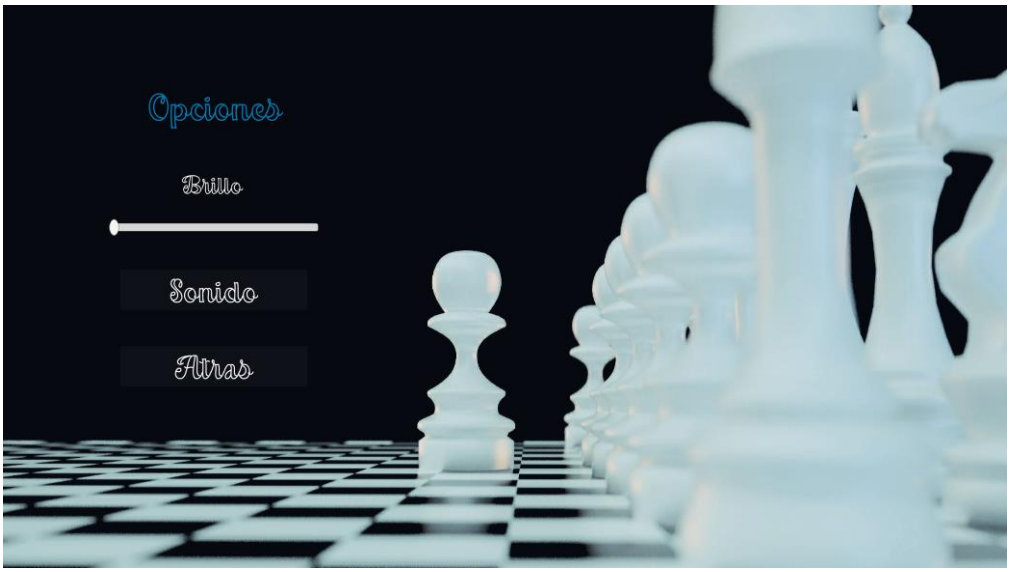
Amaury Francisco Santos Rosario 2-17-1379

República Dominicana, Santiago de los caballeros,

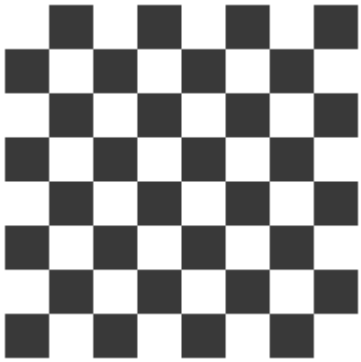
15 de agosto, 2021.

**3.1 Capturas de la Aplicación (Documentación completa del desarrollo, Scripts, Sprites, Prefabs e imágenes)**

## Menús

**Sprites.**



**Juego.**

## Scripts

**Menú Scripts.**

```
public class play : MonoBehaviour
{

    void Start()
    {

    }


    void Update()
    {

    }

    public void empezarjuego()
    {

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex +
 1);
    }


    public void CerrarJuego (){

        Application.Quit();
        Debug.Log("Salir");
    }
}
```

**Chessman Script.**

```
public class Chessman : MonoBehaviour
{

    public GameObject controller;
    public GameObject movePlate;
```

```
    private int xBoard = -1;
    private int yBoard = -1;


    private string player;


    public Sprite black_queen, black_knight, black_bishop, black_king, bl
ack_rook, black_pawn;
    public Sprite white_queen, white_knight, white_bishop, white_king, wh
ite_rook, white_pawn;

    public void Activate()
    {

        controller = GameObject.FindGameObjectWithTag("GameController");


        SetCoords();

        switch (this.name)
        {
            case "black_queen": this.GetComponent<SpriteRenderer>().sprit
e = black_queen; player = "black"; break;
            case "black_knight": this.GetComponent<SpriteRenderer>().spri
te = black_knight; player = "black"; break;
            case "black_bishop": this.GetComponent<SpriteRenderer>().spri
te = black_bishop; player = "black"; break;
            case "black_king": this.GetComponent<SpriteRenderer>().sprite
 = black_king; player = "black"; break;
            case "black_rook": this.GetComponent<SpriteRenderer>().sprite
 = black_rook; player = "black"; break;
            case "black_pawn": this.GetComponent<SpriteRenderer>().sprite
 = black_pawn; player = "black"; break;
            case "white_queen": this.GetComponent<SpriteRenderer>().sprit
e = white_queen; player = "white"; break;
            case "white_knight": this.GetComponent<SpriteRenderer>().spri
te = white_knight; player = "white"; break;
            case "white_bishop": this.GetComponent<SpriteRenderer>().spri
te = white_bishop; player = "white"; break;
            case "white_king": this.GetComponent<SpriteRenderer>().sprite
 = white_king; player = "white"; break;
            case "white_rook": this.GetComponent<SpriteRenderer>().sprite
 = white_rook; player = "white"; break;
            case "white_pawn": this.GetComponent<SpriteRenderer>().sprite
 = white_pawn; player = "white"; break;
        }
```

```
    }

    public void SetCoords()
    {

        float x = xBoard;
        float y = yBoard;


        x *= 0.66f;
        y *= 0.66f;


        x += -2.3f;
        y += -2.3f;


        this.transform.position = new Vector3(x, y, -1.0f);
    }

    public int GetXBoard()
    {
        return xBoard;
    }

    public int GetYBoard()
    {
        return yBoard;
    }

    public void SetXBoard(int x)
    {
        xBoard = x;
    }

    public void SetYBoard(int y)
    {
        yBoard = y;
    }

    private void OnMouseUp()
    {
        if (!controller.GetComponent<Game>().IsGameOver() && controller.G
etComponent<Game>().GetCurrentPlayer() == player)
        {

            DestroyMovePlates();
```

```
            InitiateMovePlates();
        }
    }

    public void DestroyMovePlates()
    {

        GameObject[] movePlates = GameObject.FindGameObjectsWithTag("Move
Plate");
        for (int i = 0; i < movePlates.Length; i++)
        {

        }
    }

    public void InitiateMovePlates()
    {
        switch (this.name)
        {
            case "black_queen":
            case "white_queen":
                LineMovePlate(1, 0);
                LineMovePlate(0, 1);
                LineMovePlate(1, 1);
                LineMovePlate(-1, 0);
                LineMovePlate(0, -1);
                LineMovePlate(-1, -1);
                LineMovePlate(-1, 1);
                LineMovePlate(1, -1);
                break;
            case "black_knight":
            case "white_knight":
                LMovePlate();
                break;
            case "black_bishop":
            case "white_bishop":
                LineMovePlate(1, 1);
                LineMovePlate(1, -1);
                LineMovePlate(-1, 1);
                LineMovePlate(-1, -1);
                break;
            case "black_king":
            case "white_king":
                SurroundMovePlate();
                break;
            case "black_rook":
            case "white_rook":
                LineMovePlate(1, 0);
                LineMovePlate(0, 1);
```

```
                LineMovePlate(-1, 0);
                LineMovePlate(0, -1);
                break;
            case "black_pawn":
                PawnMovePlate(xBoard, yBoard - 1);
                break;
            case "white_pawn":
                PawnMovePlate(xBoard, yBoard + 1);
                break;
        }
    }

    public void LineMovePlate(int xIncrement, int yIncrement)
    {
        Game sc = controller.GetComponent<Game>();

        int x = xBoard + xIncrement;
        int y = yBoard + yIncrement;

        while (sc.PositionOnBoard(x, y) && sc.GetPosition(x, y) == null)
        {
            MovePlateSpawn(x, y);
            x += xIncrement;
            y += yIncrement;
        }

        if (sc.PositionOnBoard(x, y) && sc.GetPosition(x, y).GetComponent
<Chessman>().player != player)
        {
            MovePlateAttackSpawn(x, y);
        }
    }

    public void LMovePlate()
    {
        PointMovePlate(xBoard + 1, yBoard + 2);
        PointMovePlate(xBoard - 1, yBoard + 2);
        PointMovePlate(xBoard + 2, yBoard + 1);
        PointMovePlate(xBoard + 2, yBoard - 1);
        PointMovePlate(xBoard + 1, yBoard - 2);
        PointMovePlate(xBoard - 1, yBoard - 2);
        PointMovePlate(xBoard - 2, yBoard + 1);
        PointMovePlate(xBoard - 2, yBoard - 1);
    }

    public void SurroundMovePlate()
    {
        PointMovePlate(xBoard, yBoard + 1);
        PointMovePlate(xBoard, yBoard - 1);
```

```
        PointMovePlate(xBoard - 1, yBoard + 0);
        PointMovePlate(xBoard - 1, yBoard - 1);
        PointMovePlate(xBoard - 1, yBoard + 1);
        PointMovePlate(xBoard + 1, yBoard + 0);
        PointMovePlate(xBoard + 1, yBoard - 1);
        PointMovePlate(xBoard + 1, yBoard + 1);
    }

    public void PointMovePlate(int x, int y)
    {
        Game sc = controller.GetComponent<Game>();
        if (sc.PositionOnBoard(x, y))
        {
            GameObject cp = sc.GetPosition(x, y);

            if (cp == null)
            {
                MovePlateSpawn(x, y);
            }
            else if (cp.GetComponent<Chessman>().player != player)
            {
                MovePlateAttackSpawn(x, y);
            }
        }
    }

    public void PawnMovePlate(int x, int y)
    {
        Game sc = controller.GetComponent<Game>();
        if (sc.PositionOnBoard(x, y))
        {
            if (sc.GetPosition(x, y) == null)
            {
                MovePlateSpawn(x, y);
            }

            if (sc.PositionOnBoard(x + 1, y) && sc.GetPosition(x + 1, y)
!= null && sc.GetPosition(x + 1, y).GetComponent<Chessman>().player != pl
ayer)
            {
                MovePlateAttackSpawn(x + 1, y);
            }

            if (sc.PositionOnBoard(x - 1, y) && sc.GetPosition(x - 1, y)
!= null && sc.GetPosition(x - 1, y).GetComponent<Chessman>().player != pl
ayer)
            {
                MovePlateAttackSpawn(x - 1, y);
            }
```

```
        }
    }

    public void MovePlateSpawn(int matrixX, int matrixY)
    {

        float x = matrixX;
        float y = matrixY;


        x *= 0.66f;
        y *= 0.66f;


        x += -2.3f;
        y += -2.3f;


        GameObject mp = Instantiate(movePlate, new Vector3(x, y, -
3.0f), Quaternion.identity);

        MovePlate mpScript = mp.GetComponent<MovePlate>();
        mpScript.SetReference(gameObject);
        mpScript.SetCoords(matrixX, matrixY);
    }

    public void MovePlateAttackSpawn(int matrixX, int matrixY)
    {

        float x = matrixX;
        float y = matrixY;


        x *= 0.66f;
        y *= 0.66f;


        x += -2.3f;
        y += -2.3f;


        GameObject mp = Instantiate(movePlate, new Vector3(x, y, -
3.0f), Quaternion.identity);

        MovePlate mpScript = mp.GetComponent<MovePlate>();
        mpScript.attack = true;
        mpScript.SetReference(gameObject);
        mpScript.SetCoords(matrixX, matrixY);
    }
```

```
}
```

**Game Scripts.**

```
   public GameObject piezas;


    private GameObject[,] positions = new GameObject[8, 8];
    private GameObject[] playerBlack = new GameObject[16];
    private GameObject[] playerWhite = new GameObject[16];


    private string currentPlayer = "white";


    private bool gameOver = false;


    public void Start()
    {
        playerWhite = new GameObject[] { Create("white_rook", 0, 0), Crea
te("white_knight", 1, 0),
            Create("white_bishop", 2, 0), Create("white_queen", 3, 0), Cr
eate("white_king", 4, 0),
            Create("white_bishop", 5, 0), Create("white_knight", 6, 0), C
reate("white_rook", 7, 0),
            Create("white_pawn", 0, 1), Create("white_pawn", 1, 1), Creat
e("white_pawn", 2, 1),
            Create("white_pawn", 3, 1), Create("white_pawn", 4, 1), Creat
e("white_pawn", 5, 1),
            Create("white_pawn", 6, 1), Create("white_pawn", 7, 1) };
        playerBlack = new GameObject[] { Create("black_rook", 0, 7), Crea
te("black_knight",1,7),
            Create("black_bishop",2,7), Create("black_queen",3,7), Create
("black_king",4,7),
            Create("black_bishop",5,7), Create("black_knight",6,7), Creat
e("black_rook",7,7),
            Create("black_pawn", 0, 6), Create("black_pawn", 1, 6), Creat
e("black_pawn", 2, 6),
            Create("black_pawn", 3, 6), Create("black_pawn", 4, 6), Creat
e("black_pawn", 5, 6),
            Create("black_pawn", 6, 6), Create("black_pawn", 7, 6) };

        for (int i = 0; i < playerBlack.Length; i++)
        {
            SetPosition(playerBlack[i]);
```

```
            SetPosition(playerWhite[i]);
        }
    }

    public GameObject Create(string name, int x, int y)
    {
        GameObject obj = Instantiate(piezas, new Vector3(0, 0, -
1), Quaternion.identity);
        Chessman cm = obj.GetComponent<Chessman>();
        cm.name = name;
        cm.SetXBoard(x);
        cm.SetYBoard(y);
        cm.Activate();
        return obj;
    }

    public void SetPosition(GameObject obj)
    {
        Chessman cm = obj.GetComponent<Chessman>();


        positions[cm.GetXBoard(), cm.GetYBoard()] = obj;
    }

    public void SetPositionEmpty(int x, int y)
    {
        positions[x, y] = null;
    }

    public GameObject GetPosition(int x, int y)
    {
        return positions[x, y];
    }

    public bool PositionOnBoard(int x, int y)
    {
        if (x < 0 || y < 0 || x >= positions.GetLength(0) || y >= positio
ns.GetLength(1)) return false;
        return true;
    }

    public string GetCurrentPlayer()
    {
        return currentPlayer;
    }

    public bool IsGameOver()
    {
        return gameOver;
```

```
    }

    public void NextTurn()
    {
        if (currentPlayer == "white")
        {
            currentPlayer = "black";
        }
        else
        {
            currentPlayer = "white";
        }
    }

    public void Update()
    {
        if (gameOver == true && Input.GetMouseButtonDown(0))
        {
            gameOver = false;


            SceneManager.LoadScene("Game");
        }
    }

    public void Winner(string playerWinner)
    {
        gameOver = true;


        GameObject.FindGameObjectWithTag("WinnerText").GetComponent<Text>
().enabled = true;
        GameObject.FindGameObjectWithTag("WinnerText").GetComponent<Text>
().text = playerWinner + " es el ganador";

        GameObject.FindGameObjectWithTag("RestartText").GetComponent<Text
>().enabled = true;
    }
}
```

**MovePlate Scripts.**

```
public class MovePlate : MonoBehaviour
{

    public GameObject controller;


    GameObject reference = null;


    int matrixX;
    int matrixY;


    public bool attack = false;

    public void Start()
    {
        if (attack)
        {

            gameObject.GetComponent<SpriteRenderer>().color = new Color(1
.0f, 0.0f, 0.0f, 1.0f);
        }
    }

    public void OnMouseUp()
    {
        controller = GameObject.FindGameObjectWithTag("GameController");


        if (attack)
        {
            GameObject cp = controller.GetComponent<Game>().GetPosition(m
atrixX, matrixY);

            if (cp.name == "white_king") controller.GetComponent<Game>().
Winner("black");
            if (cp.name == "black_king") controller.GetComponent<Game>().
Winner("white");

            Destroy(cp);
        }


        controller.GetComponent<Game>().SetPositionEmpty(reference.GetCom
ponent<Chessman>().GetXBoard(),
            reference.GetComponent<Chessman>().GetYBoard());
```

```
        reference.GetComponent<Chessman>().SetXBoard(matrixX);
        reference.GetComponent<Chessman>().SetYBoard(matrixY);
        reference.GetComponent<Chessman>().SetCoords();


        controller.GetComponent<Game>().SetPosition(reference);


        controller.GetComponent<Game>().NextTurn();


        reference.GetComponent<Chessman>().DestroyMovePlates();
    }

    public void SetCoords(int x, int y)
    {
        matrixX = x;
        matrixY = y;
    }

    public void SetReference(GameObject obj)
    {
        reference = obj;
    }

    public GameObject GetReference()
    {
        return reference;
    }
}
```

**3.2 Prototipos**

Un prototipo es su forma de probar conceptos sin invertir demasiado tiempo o esfuerzo. Puede imaginar lo beneficioso que es descubrir que una idea no funcionará antes de dedicar tiempo.

la creación de prototipos también es útil para encontrar formas de modificar ciertas mecánicas del juego y descubrir cómo hacerlo lo más divertido posible. Incluso si varias ideas terminan en fracaso en la etapa de prototipo, En cuanto al juego de ajedrez hay muchos prototipos en línea en el que nos podemos guiar para resolver cualquier problema.

**3.3 Perfiles de Usuarios**

Se requiere un perfil de usuario cuando se juega un juego en línea o sin conexión. Su puntuación de jugador, logros y progreso del juego solo se pueden ver ahí. Yo y mi compañero trataremos de implementar este perfil mas adelante en el juego.

### 3.4 Usabilidad

El juego es fácil de comprender, solo se le dará un click en las piezas que se desea mover y esta presentara los movimientos que puede hacer, en caso de que este bloqueada esta cortara el camino a recorrer. Antes de comenzar el juego podrás elegir entre jugar con otro jugador en el mismo pc u contra la maquina en diferentes modos.

### 3.5 Test

Se probará que cada regla del juego se cumpla, como es el salto del caballo a otras piezas, que el peón pueda dar 2 paso en el primer movimiento o que el peón al llegar al final de la tabla se cambie por otra pieza que allá sido eliminada.

Después de probar las funciones, se pondrá a tercero a probar el juego y encontrar posible fallos o bug. Este proceso durara hasta que el juego se abandone.

### 3.6 Versiones de la Aplicación

Alfa es la prueba donde los desarrolladores probaremos los posibles fallos y solucionaremos. Esta prueba es más específica y buscan errores comunes.

Beta es la prueba donde podremos a los terceros a jugar y ver los posibles fallos para encontrarle solucione. Esta prueba busca encontrar posibles fallos que pasaron por alto en el momento de desarrollarse.

https://github.com/amaurysantos27/capitulo2

https://github.com/VM-Alexander/Proyecto_Final