UNIVERSIDAD TECNOLÓGICA DE SANTIAGO (UTESA)

SISTEMA CORPORATIVO

Facultad de Arquitectura e Ingeniería, carrera de Ingeniería en Sistemas Computacionales

Programación de Videojuegos

Capítulo 3 – Proyecto Final

Presentado a:
Iván Mendoza

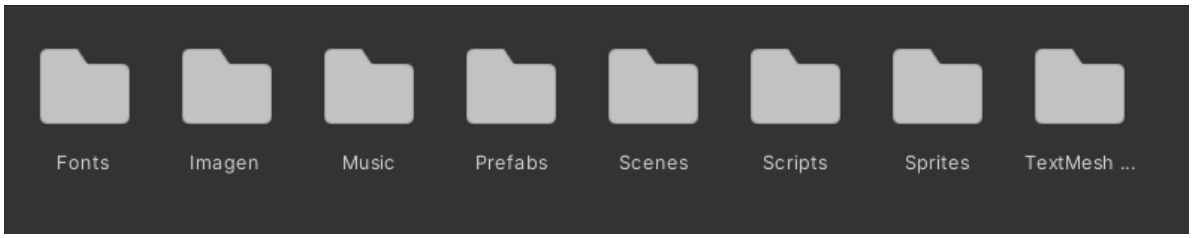
Presentado por:

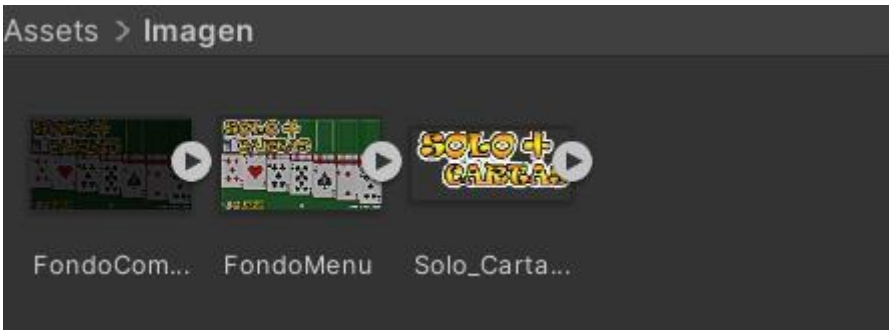
Wilber Salcedo Mata	2-18-0414
Robert Junior Álvarez	2-16-0738
Jerinel Mendoza	1-17-0996

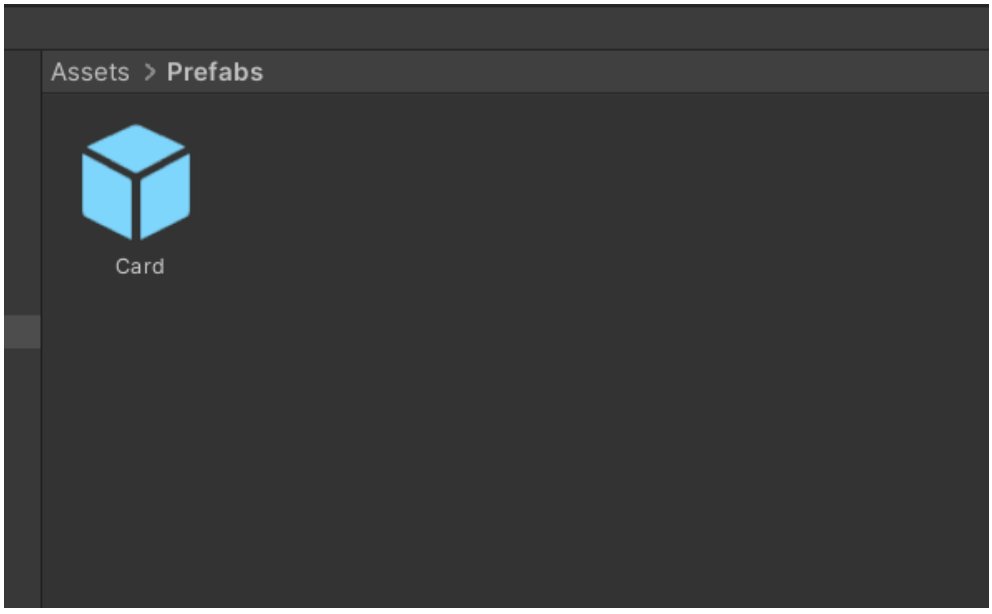
14 de diciembre del 2022
Santiago de los Caballeros,
República Dominicana.

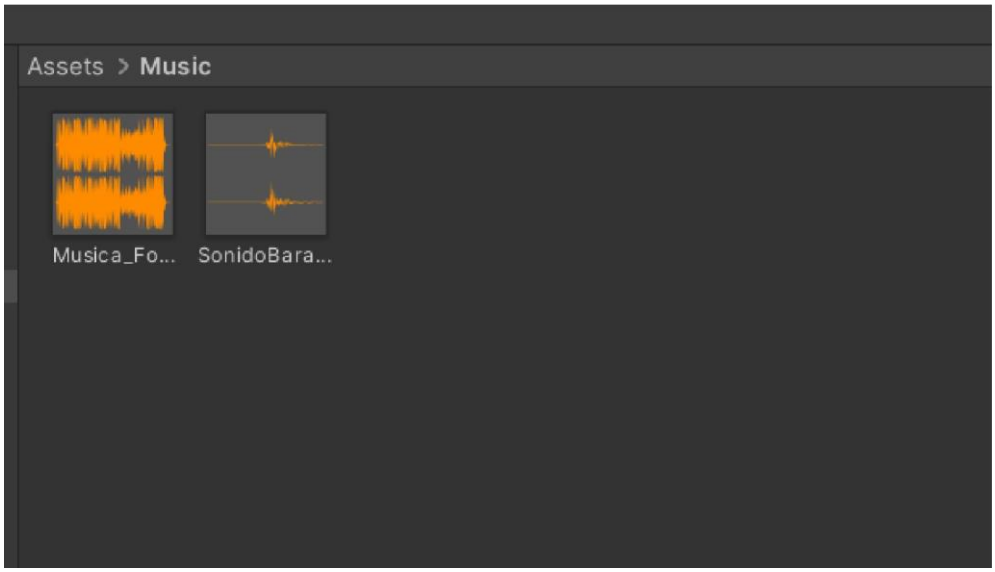
3.1 Capturas de la Aplicación Scripts, Sprites, Prefabs e imágenes)

- **Capturas de la aplicación**

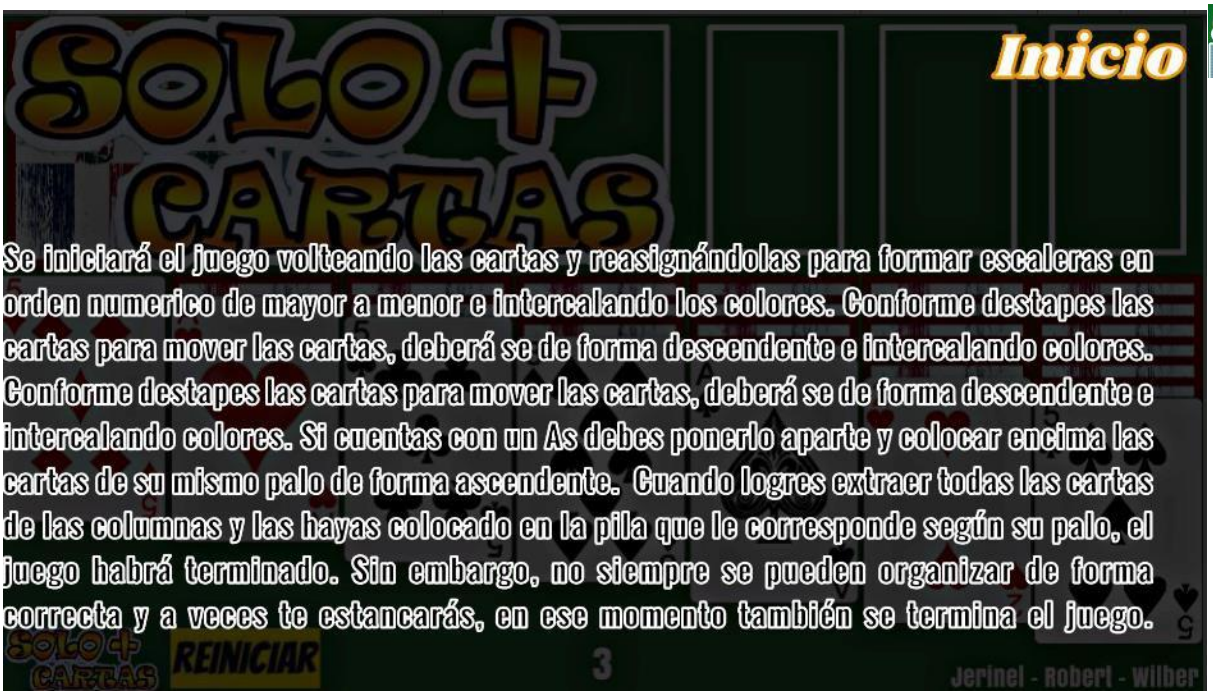


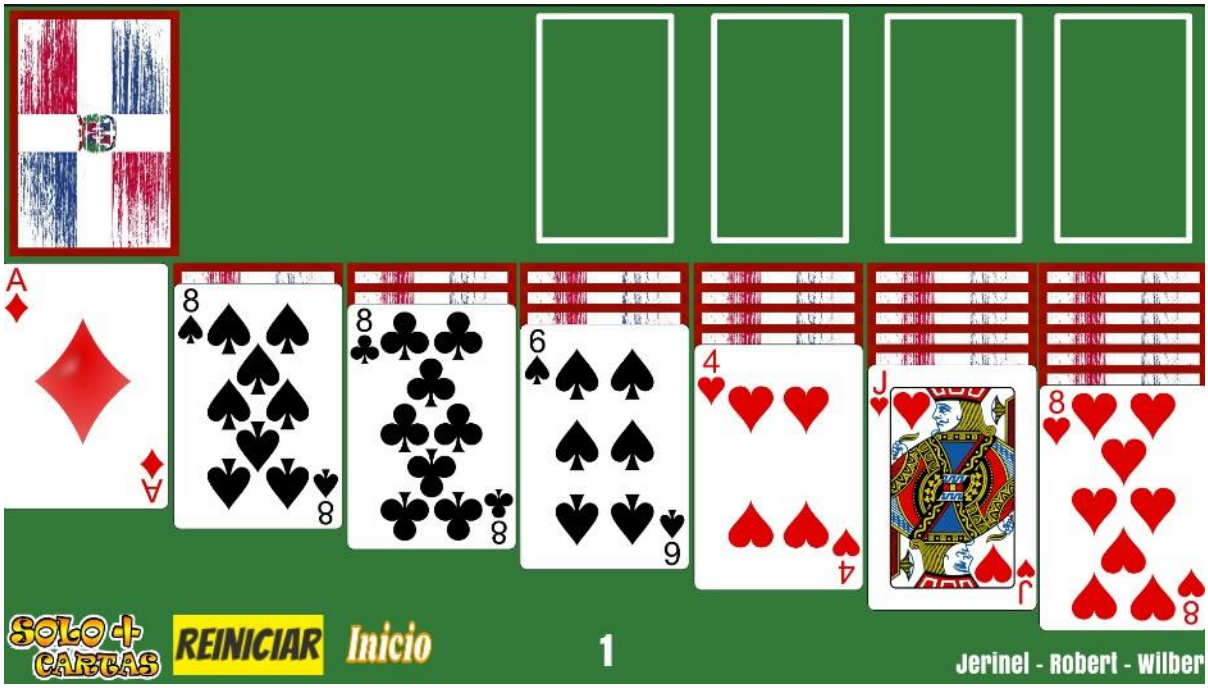





Assets > Music
Musica_Fo...
SonidoBara...


SOLO 4 CARTAS
Jugar
Como jugar
Salir
SOLO 4 CARTAS
REINICIAR
3
Jerinel - Robert - Wilber





Descripción de scripts:

- **Selectable:** En esta parte se hacen todas las referencias a las ejecuciones de usabilidad del juego, partiendo desde el área donde se hace la acción mediante el script UserInput.
- **Solitaire:** Este script abarca toda la lógica del jeugo con sus atributos generales, reglas, selecciones de acciones, mismas que se ejecutan conforme el jugador va tomando ciertas decisiones a lo largo del gameplay.
- **UpdateSprite:** Esta parte se encarga de actualizar la interfaz de usuario, brindando todas las salidas gráficas de actualizaciones para ir avanzando el la partida.
- **UserInput:** Este script abarca toda la parte de las interacciones del usuario, con el juego. Vela por ver las ejecuciones dadas por el mismo, dependiendo lo que este decida.

- **MenuInicial:** Este script sirve para cambiar de escenas dentro del juego

Selectable script:

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

public class Selectable : MonoBehaviour
{
    public bool top = false;
    public string suit;
    public int value;
    public int row;
    public bool faceUp = false;
    public bool inDeckPile = false;


    private string valueString;

    // Start is called before the first frame update
    void Start()
    {
        if(CompareTag("Card"))
        {
            suit = transform.name[0].ToString();

            for (int i = 1; i < transform.name.Length;
i++)
            {
            char c = transform.name[i];
            valueString = valueString + c.ToString();
            }
```

```
            if(valueString == "A")
            {
                value =1 ;
            }
            if(valueString == "2")
            {
                value = 2;
            }
            if(valueString == "3")
            {
                value =3 ;
            }
            if(valueString == "4")
            {
                value = 4;
            }
            if(valueString == "5")
            {
                value = 5;
            }
            if(valueString == "6")
            {
                value = 6;
            }
            if(valueString == "7")
            {
                value = 7;
            }
            if(valueString == "8")
            {
                value = 8;
            }
            if(valueString == "9")
            {
```

```
                value = 9;
            }
            if(valueString == "10")
            {
                value = 10;
            }
            if(valueString == "J")
            {
                value = 11;
            }
            if(valueString == "Q")
            {
                value = 12;
            }
            if(valueString == "K")
            {
                value = 13;
            }



        }

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Solitaire script:

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

public class Solitaire : MonoBehaviour
{
    public TextMeshProUGUI temporizador;

    public Sprite[] cardFaces;
    public GameObject cardPrefab;
    public GameObject deckButton;
    public GameObject[] bottomPos;
    public GameObject[] topPos;
    public List<string> tripsOnDisplay = new
List<string>();
    public List<List<string>> deckTrips = new
List<List<string>>();


    public static string[] suits = new string[] { "C",
"D", "H", "S" };
    public static string[] values = new string[] { "A",
"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q",
"K" };
    public List<string>[] bottoms;
    public List<string>[] tops;

    private List<string> bottom0 = new List<string>();
    private List<string> bottom1 = new List<string>();
```

```
    private List<string> bottom2 = new List<string>();
    private List<string> bottom3 = new List<string>();
    private List<string> bottom4 = new List<string>();
    private List<string> bottom5 = new List<string>();
    private List<string> bottom6 = new List<string>();


    public List<string> deck;
    public List<string> discardPile = new
List<string>();
    private int trips;
    private int tripsRemainder;
    private int deckLocation;
    private float TimeT=0;



    // Start is called before the first frame update
    void Start()
    {
        bottoms = new List<string>[] { bottom0,
bottom1, bottom2, bottom3, bottom4, bottom5, bottom6 };
        PlayCards();
    }

    // Update is called once per frame
    void Update()
    {
        TimeT+=Time.deltaTime;

        temporizador.text = ""+ TimeT.ToString("0");
    }

    public void PlayCards()
```

```
    {
        deck = GenerateDeck();
        Shuffle(deck);

        foreach(string card in deck)
        {
            print(card);
        }

        SolitaireSort();
        StartCoroutine(SolitaireDeal());
        SortDeckIntoTrips();
    }

     public static List<string> GenerateDeck()
    {
        List<string> newDeck = new List<string>();
        foreach (string s in suits)
        {
            foreach (string v in values)
            {
                newDeck.Add(s + v);
            }
        }
        return newDeck;
    }

    void Shuffle<T>(List<T> list)
    {
        System.Random random = new System.Random();
        int n = list.Count;
        while (n > 1)
        {
            int k = random.Next(n);
```

```
            n--;
            T temp = list[k];
            list[k] = list[n];
            list[n] = temp;
        }
    }

    IEnumerator SolitaireDeal()
    {
        for(int i = 0; i < 7; i++)
        {

            float yOffset = 0;
            float zOffset = 0.03f;
            foreach(string card in bottoms[i])
            {
                yield return new WaitForSeconds(0.01f);
                GameObject newCard =
Instantiate(cardPrefab, new
Vector3(bottomPos[i].transform.position.x,
bottomPos[i].transform.position.y - yOffset,
bottomPos[i].transform.position.z - zOffset),
Quaternion.identity, bottomPos[i].transform);
                newCard.name = card;
                newCard.GetComponent<Selectable>().row
= i;
                if(card == bottoms[i][bottoms[i].Count
- 1])
                {

newCard.GetComponent<Selectable>().faceUp = true;
                }
```

```
                yOffset = yOffset + 0.3f;
                zOffset = zOffset + 0.03f;
                discardPile.Add(card);
            }
        }

         foreach(string card in discardPile)
        {
            if(deck.Contains(card))
            {
                deck.Remove(card);
            }
        }
        discardPile.Clear();
    }

    void SolitaireSort()
    {
       for(int i = 0; i < 7; i++)
       {
         for(int j = i; j < 7; j++)
         {
            bottoms[j].Add(deck.Last<string>());
            deck.RemoveAt(deck.Count - 1);
         }
       }
    }


  public void SortDeckIntoTrips()
  {
        trips = deck.Count / 3;
        tripsRemainder = deck.Count % 3;
        deckTrips.Clear();
```

```
        int modifier = 0;
        for(int i = 0; i < trips; i++)
        {

            List<string> myTrips = new List<string>();
            for(int j = 0; j < 3; j++)
            {
                myTrips.Add(deck[j + modifier]);
            }
            deckTrips.Add(myTrips);
            modifier = modifier + 3;
        }

            if(tripsRemainder!=0)
            {

                List<string> myRemainders = new
List<string>();
                modifier = 0;

                for(int k=0; k < tripsRemainder; k++)
                {

                    myRemainders.Add(deck[deck.Count -
tripsRemainder + modifier]);
                    modifier++;
                }

                deckTrips.Add(myRemainders);
                trips++;
            }
            deckLocation = 0;
    }
```

```
      public void DealFromDeck()
      {


        // add remainings card to discard pile
        foreach(Transform child in
deckButton.transform)
        {
          if(child.CompareTag("Card"))
          {
            deck.Remove(child.name);
            discardPile.Add(child.name);
            Destroy(child.gameObject);
          }
        }


         if(deckLocation < trips)
         {
            // draw 3 new cards
            tripsOnDisplay.Clear();
            float xoffset = 2.5f;
            float zoffset = -0.2f;
            foreach(string card in
deckTrips[deckLocation])
            {
               GameObject newTopCard =
Instantiate(cardPrefab, new
Vector3(deckButton.transform.position.x + xoffset,
deckButton.transform.position.y,deckButton.transform.po
sition.z + zoffset), Quaternion.identity,
deckButton.transform);
```

```
                xoffset = xoffset + 0.5f;
                zoffset = zoffset - 0.2f;
                newTopCard.name = card;
                tripsOnDisplay.Add(card);

newTopCard.GetComponent<Selectable>().faceUp = true;

newTopCard.GetComponent<Selectable>().inDeckPile =
true;

            }
            deckLocation++;
        }
        else
        {

            //RestackTopDeck
            RestackTopDeck();
        }


    }

    void RestackTopDeck()
    {
        foreach(string card in discardPile)
      {
            deck.Add(card);
      }
      discardPile.Clear();
      SortDeckIntoTrips();

    }
```

```
}
```

UpdateSprite:

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

public class UpdateSprite : MonoBehaviour
{
    public Sprite cardFace;
    public Sprite cardBack;
    private SpriteRenderer spriteRenderer;
    private  Selectable  selectable;
    private Solitaire solitaire;
    private UserInput userInput;



    // Start is called before the first frame update
    void Start()
    {
        List<string> deck = Solitaire.GenerateDeck();
        solitaire = FindObjectOfType<Solitaire>();
        userInput = FindObjectOfType<UserInput>();

        int i = 0;
        foreach (string card in deck)
        {
            if (this.name == card)
            {
                cardFace = solitaire.cardFaces[i];
                break;
```

```
            }
            i++;
        }
        spriteRenderer =
GetComponent<SpriteRenderer>();
        selectable = GetComponent<Selectable>();
    }

    // Update is called once per frame
    void Update()
    {

        if (selectable.faceUp == true)
        {
            spriteRenderer.sprite = cardFace;
        }
        else
        {
            spriteRenderer.sprite = cardBack;
        }

        if(userInput.slot1)
        {
            if(name == userInput.slot1.name)
            {
                spriteRenderer.color = Color.yellow;
            }
            else
            {
                spriteRenderer.color = Color.white;
            }

        }
```

```
    }
}
```

UserInput:

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

public class UserInput : MonoBehaviour
{

   public AudioSource AudioSource;
   public GameObject slot1;
   private Solitaire solitaire;
    // Start is called before the first frame update
    void Start()
    {
        solitaire = FindObjectOfType<Solitaire>();
        slot1 = this.gameObject;
    }

    // Update is called once per frame
    void Update()
    {
        GetMouseClick();
    }

    void GetMouseClick(){

        if(Input.GetMouseButtonDown(0)){
```

```
            Vector3 mousePosition =
Camera.main.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, -
10));

            RaycastHit2D hit =
Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Input.
mousePosition), Vector2.zero);

            if(hit){
                // what has been hit? Deck//Card/
Emptyslot..
                if(hit.collider.CompareTag("Deck")){
                //clicked deck
                Deck();
                }
                else
if(hit.collider.CompareTag("Card")){
                //clicked Card
                    Card(hit.collider.gameObject);
                }
                else
if(hit.collider.CompareTag("Top")){
                //clicked Top
                    Top();
                }
                else
if(hit.collider.CompareTag("Bottom")){
                //clicked Bottom
                    Bottom();
                }

            }
        }
```

```
    }

        void Deck()
        {
            //Deck click actions
            print("clicked on Deck");
            AudioSource.Play();
            solitaire.DealFromDeck();
        }

        void Card(GameObject selected)
        {

            //Card click actions
            print("clicked on Card");

            if(slot1 == this.gameObject)
            {
                slot1 = selected;
            }
            else if(slot1 != selected)
            {

                if(Stackable(selected))
                {
                    //Stackit
                }
                else
                {

                  slot1 = selected;
                }

            }
```

```
        }

        void Top()
        {
            //Top click actions
            print("clicked on Top");

        }

        void Bottom()
        {
            //Bottom click actions
            print("clicked on Bottom");

        }

    bool Stackable(GameObject selected)
    {
        Selectable s1 =
slot1.GetComponent<Selectable>();
        Selectable s2 =
selected.GetComponent<Selectable>();


        if(s2.top)
        {
            if (s1.suit == s2.suit || (s1.value == 1 &&
s2.suit == null))
            {
                if(s1.value == s2.value + 1)
                {

                    return true;
```

```
            }
        }
        else
        {
            return false;
        }
    }
    else
    {
        if(s1.value == s2.value - 1)
        {
            bool card1Red = true;
            bool card2Red = true;

            if (s1.suit == "C" || s1.suit == "S")
            {

             card1Red = false;

            }

            if (s2.suit == "C" || s2.suit == "S")
            {

             card2Red = false;

            }
            if(card1Red == card2Red)
            {
                print("Not stackable");
                return false;
            }
            else
```

```
                {
                    print("Stackable");
                    return true;
                }

            }

        }

        return false;
    }

}
```

```
void Win()
{
    highScorePanel.SetActive(true);
}
```

MenuInicial:

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

public class MenuInicial : MonoBehaviour
{
    // Start is called before the first frame update
    public void Jugar()
    {
        SceneManager.LoadScene("SampleScene");
    }

    public void Salir()
    {
        Debug.Log("Salir");
        Application.Quit();
    }

    public void Inicio()
    {
        SceneManager.LoadScene("MenuInicio");
    }

    public void ComoJugar()
    {
        SceneManager.LoadScene("ComoJugar");
    }
}
```

ScoreKeeper:

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

public class ScoreKeeper : MonoBehaviour
{
    public Selectable[] topStacks;
    public GameObject highScorePanel;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (HasWon())
        {
            Win();
        }
    }

    public bool HasWon()
    {
        int i = 0;
        foreach(Selectable topstack in topStacks)
        {
            i += topstack.value;
        }
        if( i >= 52)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    void Win()
    {
        highScorePanel.SetActive(true);
    }
}
```

3.2 Prototipos

Para este juego se hicieron prototipos de pruebas básicos, sin todas las reglas que incluye este en la vida real (física),

3.3 Perfiles de usuarios

El público objetivo del videojuego es para personas entre 10 años de edad en adelante. Personas con interés en sucesos históricos. Personas con interés en el género Beat ‘Em Up.

3.4 Usabilidad

- **Controles:** Sólo se utilizará el mouse para jugar (Click izquierdo)

3.5 Test

Test 1:

Individuo 1:

Sexo	Hombre
Edad	25
Nivel de estudios	Universitario

Resultados

Puntos a evaluar	**Puntuación**
Jugabilidad	**4**
Dificultad	**1**
Control de personaje	**5**
Guía de usuario	**4**
Información proporcionada por el juego	**3**
Diseño visual	**3**
Coherencia	**4**

Individuo 2:

Sexo	Mujer
Edad	21
Nivel de estudios	Universitario

Resultados

Puntos a evaluar	**Puntuación**
Jugabilidad	**4**
Dificultad	**5**
Control de personaje	**3**
Guía de usuario	**5**
Información proporcionada por el juego	**3**
Diseño visual	**2**
Coherencia	**5**

Individuo 3:

Sexo	Hombre
Edad	25
Nivel de estudios	Universitario

Resultados

Puntos a evaluar	**Puntuación**
Jugabilidad	**3**
Dificultad	**3**
Control de personaje	**5**
Guía de usuario	**2**
Información proporcionada por el juego	**3**
Diseño visual	**4**
Coherencia	**1**

Resultados test 1:

Puntos a evaluar	**Puntuación**
Jugabilidad	**11/3 = 3.7**
Dificultad	**9/3 = 3**
Control de personaje	**13/3 = 4.3**
Guía de usuario	**11/3 = 3.7**
Información proporcionada por el juego	**9/3 = 3**
Diseño visual	**9/3 = 3**
Coherencia	**10/3 = 3.3**

Test 2:

Individuo 1:

Sexo	Mujer
Edad	20
Nivel de estudios	Universitario

Resultados

Puntos a evaluar	**Puntuación**
Jugabilidad	**4**
Dificultad	**2**
Control de personaje	**1**
Guía de usuario	**5**
Información proporcionada por el juego	**3**
Diseño visual	**3**
Coherencia	**2**

Individuo 2:

Resultados

Sexo	Mujer
Edad	22
Nivel de estudios	Universitario

Puntos a evaluar	**Puntuación**
Jugabilidad	**2**
Dificultad	**3**
Control de personaje	**4**
Guía de usuario	**3**
Información proporcionada por el juego	**4**
Diseño visual	**3**
Coherencia	**1**

Resultados test 2:

Puntos a evaluar	**Puntuación**
Jugabilidad	**6/2 = 3**
Dificultad	**5/2 = 2.5**
Control de personaje	**5/2 = 2.5**
Guía de usuario	**8/2 = 4**
Información proporcionada por el juego	**7/2 = 3.5**
Diseño visual	**6/2 = 3**
Coherencia	**3/2 = 1.5**

3.6 Versiones de la aplicación

La aplicación se lanzará en la versión 1.0.0 en la cual incluirá todas las funcionalidades necesarias para jugar.

Debido a que el juego es de mesa, no se tiene previsto ninguna actualización de versión del código.

Link del juego: https://wilbersx01.itch.io/solitario-cartas
Link del Repositorio de GitHub: https://github.com/wilbersx001/Proyecto-Final