Conquis App

Documentación del juego

Universidad de Montemorelos
Facultad de Ingeniería y Tecnología

Por:
Alvin García
Michael Arthur
Samuel Pérez
Sarah Hernández

En cumplimiento a la materia de:
Desarrollo de Software para entretenimiento

Profesor.
Ing. Ignacio Cruz Domínguez

02 de diciembre de 2020. Montemorelos, Nuevo León

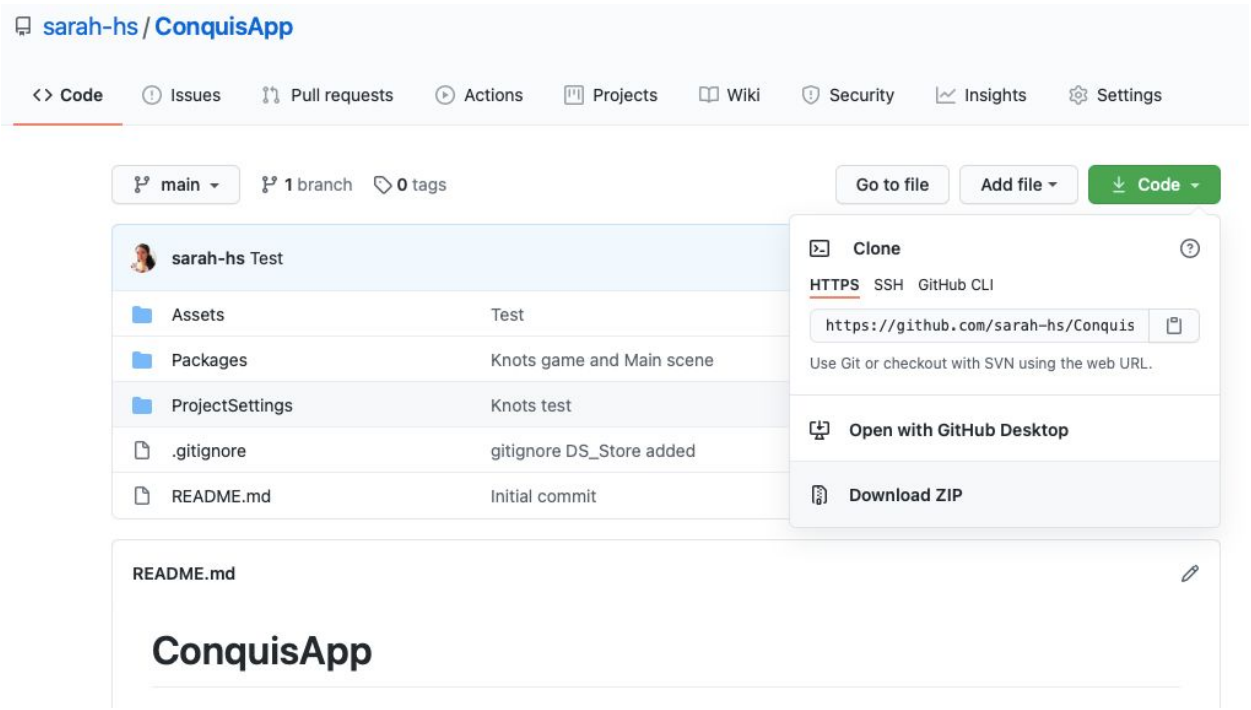
ÍNDICE DE CONTENIDO

Documentación del Juego

1. Introducción

En el siguiente enlace de github se puede ver el proyecto completo y es descargable dando click en “Download as zip” de la siguiente manera:

Enlace: https://github.com/sarah-hs/ConquisApp



Este juego consiste en una serie de juegos que ayudan a los conquistadores a completar, entender y practicar especialidades. Son aproximadamente 5 especialidades, y se pueden agregar muchísimas más. Se van acumulando puntos en cada juego y el objetivo final es completar las especialidades con el mayor puntaje posible.

1.1 Descripción

Composiciòn de la escena: Primeramente se compone de un menú general para seleccionar la especialidad a practicar, la configuración del juego y el puntaje. En

futuros trabajos estos puntajes se pueden actualizar vía internet para poder competir contra otros jugadores.
Género: cualquier público, principalmente jóvenes conquistadores o maestros que trabajan en el área de conquistadores.
Consiste en los siguientes juegos:

- Especialidad de nudos
- Memorama de personajes bíblicos
- Tic-tac-toe
- Damas inglesas
- Ajedrez

1.2 Requisitos mínimos del sistema

Plataforma: PC
SO: Windows 7/8/10
Procesador: Intel Pentium Dual Core / AMD FX
Memoria: 2 GB RAM
Tarjeta de Video: Intel HD Graphics
Almacenamiento: 100 MB de espacio libre
Tarjeta de Sonido: DirectX

2. Pantalla principal

2.1 Tutoriales usados

Utilizamos los siguientes tutoriales para crear el código de funcionamiento de la pantalla principal: https://www.youtube.com/watch?v=-SL8Zhtxb9c
https://www.youtube.com/watch?v=EdhurHPM7Gk

Añadir o quitar sonido al dar click: https://www.youtube.com/watch?v=FxedJgTsFyQ
Sonido:
https://answers.unity.com/questions/1339832/c-on-click-sound-to-play-before-loading-scene.html

https://answers.unity.com/questions/1531220/how-to-play-a-sound-when-a-prefab-button-that-chan.html

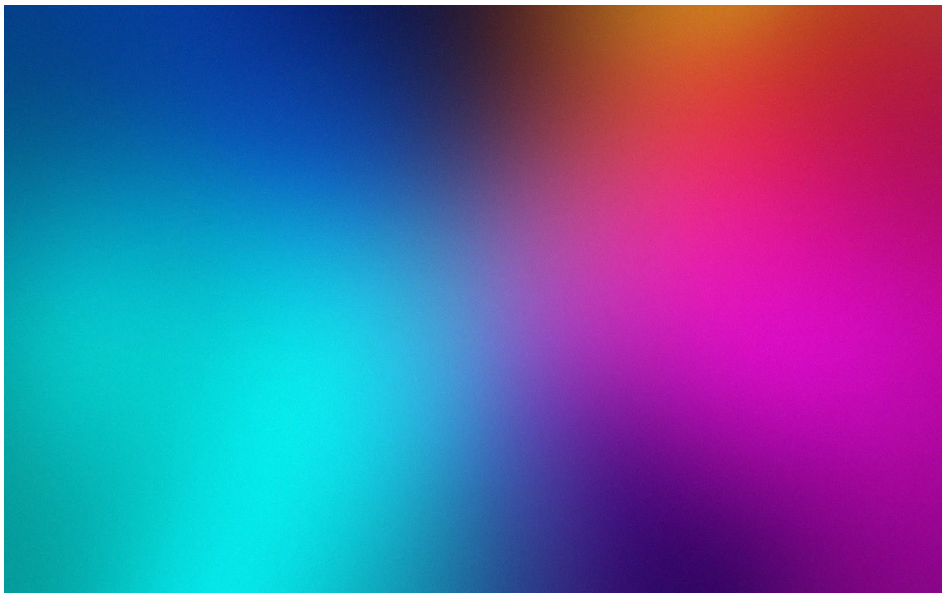
2.2 Sprites y prefabs

2.2.1 Fuentes de letra

Las fuentes de letra descargadas de FontSpace que utilizamos:
Lyons: https://www.fontspace.com/lyons-2-font-f25160
y Retronome regular: https://www.fontspace.com/retronome-font-f40089

2.2.2 Assets

Imagen de background con colores azul, rojo y morado.
https://drive.google.com/file/d/1x0EMHM4ssW4XK3yWKzuNszoRYTPsfUni/view?usp=sharing

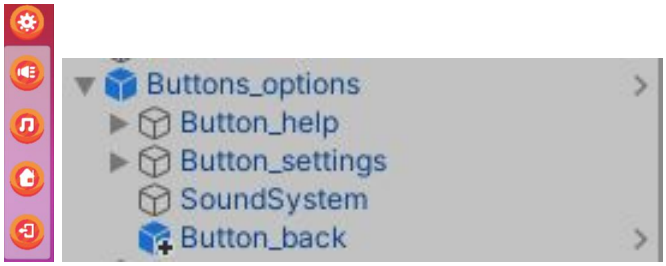


Estos son los sprites utilizados para los íconos de la App. Fueron descargados de la página de iconscout: https://iconscout.com/icon-pack/cartoon-game-ui-essential-icons.

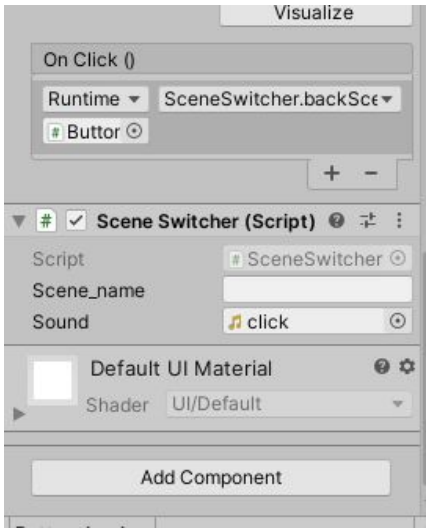
2.2.3 Prefabs

Canvas_Game

Buttons_options



Button_back



2.3 Scripts

2.3.1 Script: SceneSwitcher.cs

Este método se llama en el onClick() de los botones correspondientes a cambios de escena, Back Scene, Home y Exit Game.
Contiene los métodos loadScene(), backScene() y exitGame():

loadScene() - guarda la escena actual como prev_scene y carga una escena específica del juego dado el nombre de esa nueva escena.

backScene() - carga la escena guardada como prev_scene.

exitGame() - Cierra la aplicación.

2.3.2 Script: PanelManager.cs

Este método se implementa en el botón que abre un panel como el de settings o el de info del juego. Contiene los siguientes métodos:

openandclose() - determina si el panel está abierto y lo cierra o viceversa.

2.3.2 Script: SoundManager.cs

Se encarga de hacer mute a la música y a los sonidos del juego. Además que aplica sonido al botón de click. Contiene los siguientes métodos:

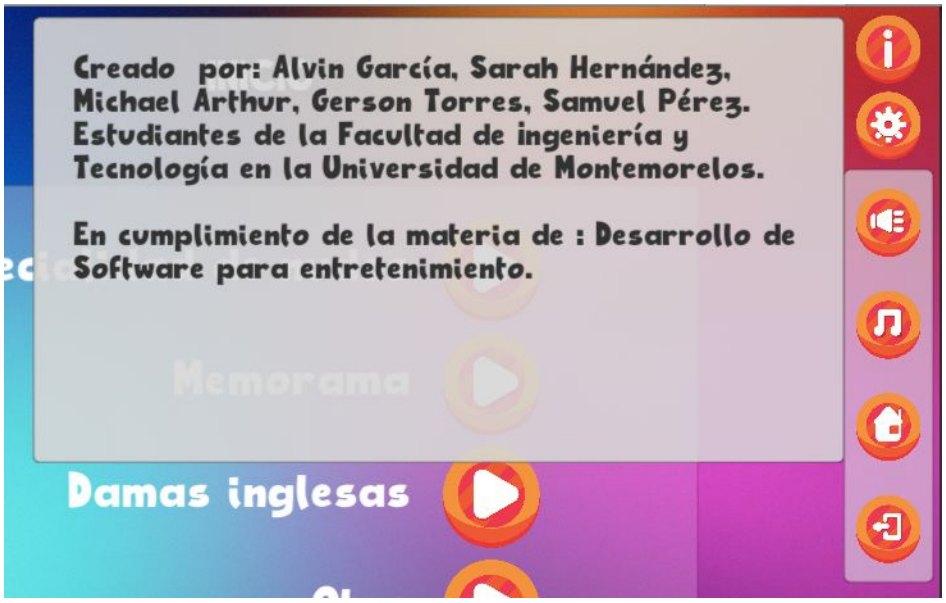
stopMusic() - determina si la música se está reproduciendo y le pone stop o play.

muteSounds() - mutea los sonidos del juego o los enciende.

playClick() - Reproduce un sonido de click al presionar un botón.

2.4 Capturas de pantalla

El ícono de información muestra la descripción general del juego.



3. Test de especialidad de nudos

3.1 Tutoriales usados

https://www.youtube.com/watch?v=eKzKntYG7Pc

https://www.youtube.com/watch?v=PDYB32qAsLU

Fuentes de información sobre nudos:

Manual de nudos del club de conquistadores:

https://drive.google.com/file/d/1b6792WbYcGiq_dCyABTPt-rSpf2jNSBC/view?usp=sharing

Imágenes de especialidades de pinterest:

https://drive.google.com/drive/folders/1aBXCXPP3lP2voTvUM7kcWG9SIfWlws6y?usp=sharing

3.2 Sprites

Background de la pantalla principal de especialidades de nudos:
https://drive.google.com/file/d/106fuVHZkDX15kpvTvdDnBtBqgO5mj2jm/view?usp=sharing

Background de la pantalla de test y la de menú de nudos:
https://drive.google.com/file/d/1EE_raU-GNYItEz7hCUOgS_gdO7n_rtU1/view?usp=sharing

3.3 Scripts

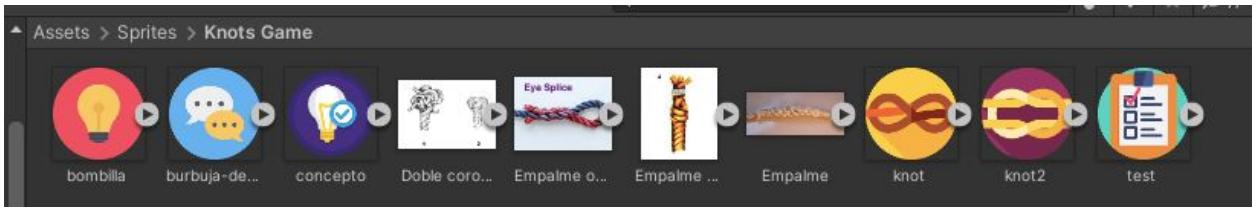
Contiene un solo script que se llama TestManager.cs. Este script se encarga de cargar las preguntas del juego en un scrollView. Las preguntas ya están diseñadas en objetos vacíos, sin embargo este script se encarga de activarlas o desactivarlas según corresponda al ir contestando el test de especialidad de nudos.
Este script contiene los métodos:
NextQuestion() - que carga la siguiente pregunta cuando es llamado con la acción del botón next y determina si ya hemos llegado a la pregunta final para entonces preguntar al usuario si desea terminar o aún no.
PrevQuestion() - carga la pregunta con el id anterior al actual y si el id es de una pregunta menor a cero (lo cual significa que no existe) entonces no carga ninguna nueva pregunta.
CancelFinished() - si el usuario decide no terminar aún el juego al llegar a la pregunta final, este método se llama y simplemente cierra el panel de preguntar al usuario si desea terminar.
YesFinished() - si el usuario desea terminar el juego entonces este método carga la escena del menú inicial.

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

public class TestManager : MonoBehaviour
```

```
{

    public Text tittle;
    public Text instruction;
    public ScrollRect scrollView;

    private string[] tils = { "1. DEFINE LOS SIGUIENTES TÉRMINOS",
                              "2. CÓMO CUIDAR LA CUERDA",
                              "3. CUERDA DE TRES FIBRAS Y CUERDA LISA (CON ALMA)",
                              "4. IDENTIFICA LAS SIGUIENTES CLASES DE CUERDA",
                              "5. LA CUERDA SINTÉTICA",
                              "6. IDENTIFICA LO SSIGUIENTES NUDOS"};

    private string[] ins = { "Relaciona cada definición con su concepto correspondiente:",
                              "Selecciona todas aquellas acciones que hacen referencia a un buen cuidado de las
cuerdas.",
                              "¿Cuáles son las diferencias entre una cuerda lisa y la de 3 fibras?",
                              "Escoge entre Manila, Sisal, Nylon y Polipropileno",
                              "¿Cuáles son algunas de las ventajas de la cuerda sintética?",
                              "Elige el nombre del nudo en cada imagen."};

    public GameObject q1;
    public GameObject q2;
    public GameObject q3;
    public GameObject q4;
    public GameObject q5;
    public GameObject q6;
    public Button prev;
    public Button next;
    public GameObject panel_terminar;
    private GameObject current_question;
    private GameObject[] questions;
    private int i = -1;
    private float score = 0;

    // Start is called before the first frame update
    void Start()
    {
        current_question = GameObject.FindGameObjectWithTag("Panel");
        questions = new GameObject[tils.Length];
        questions[0] = q1;
        questions[1] = q2;
        questions[2] = q3;
        questions[3] = q4;
        questions[4] = q5;
        questions[5] = q6;
    }

    public void NextQuestion()
    {
        if (i==-1) {
            scrollView.gameObject.SetActive(true);
            prev.gameObject.SetActive(true);
            next.gameObject.SetActive(true);
        }
        if (i+1 < tils.Length) {
```

```
            current_question.gameObject.SetActive(false);
            i += 1;
            tittle.text = tils[i];
            instruction.text = ins[i];
            current_question = questions[i];
            Debug.Log(current_question);
            current_question.gameObject.SetActive(true);
        } else {
            TestFinished();
        }
    }

    public void PrevQuestion()
    {
        if (i-1 >= 0) {
            current_question.gameObject.SetActive(false);
            i -= 1;
            tittle.text = tils[i];
            instruction.text = ins[i];
            current_question = questions[i];
            current_question.gameObject.SetActive(true);
        }
    }

    void TestFinished()
    {
        panel_terminar.gameObject.SetActive(true);
    }

    public void CancelFinished()
    {
        panel_terminar.gameObject.SetActive(false);
    }
}
```

3.4 Capturas de pantalla



4. Memorama

4.1 Tutoriales usados

A continuación se presentan los tutoriales usados para construir el juego del memorama:

Unity 5 - Juego de buscar parejas P.1 - Unity Tutorial español c#

Unity 5 - Juego de buscar parejas P.2 - Unity Tutorial español c#

Unity 5 - Juego de buscar parejas P.3 - Unity Tutorial español c#

Unity 5 - Juego de buscar parejas P.4 - Unity Tutorial español c#

4.2 Sprites

[Tarjetas del memorama]

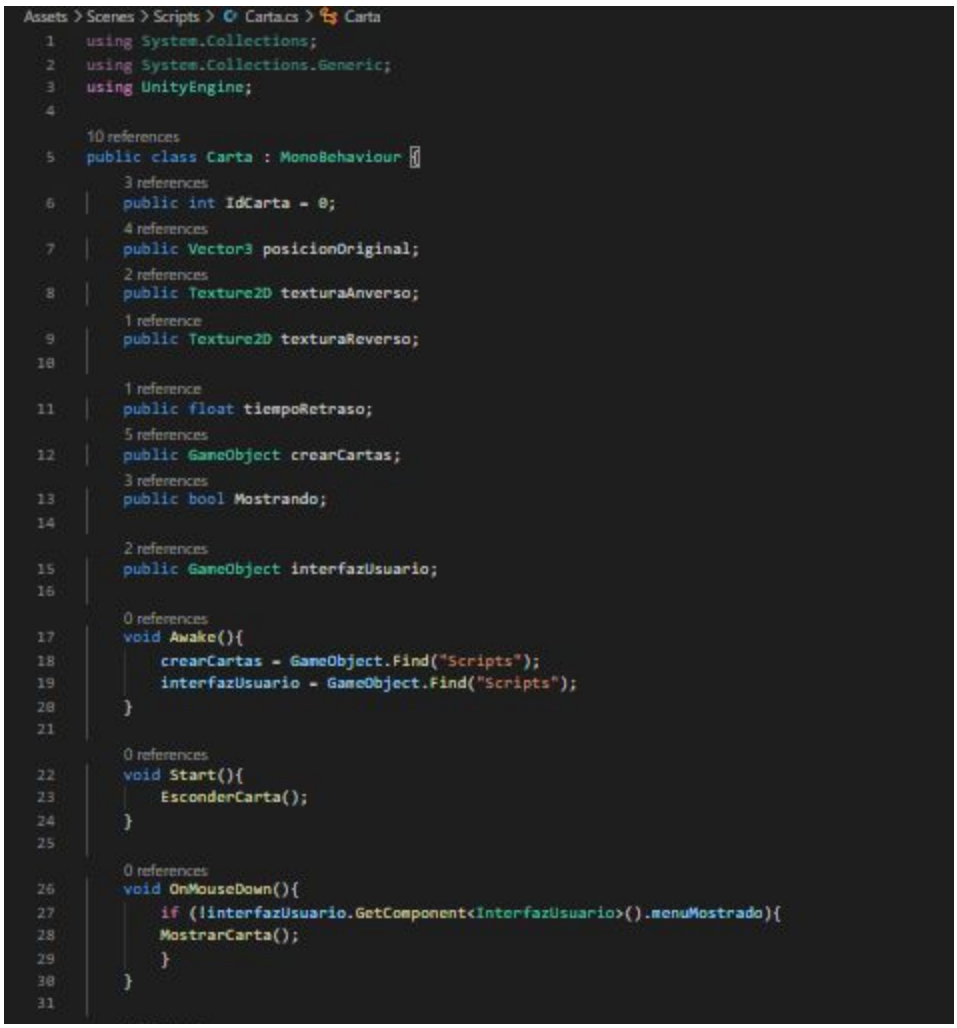
[Tarjeta destapada del memorama]

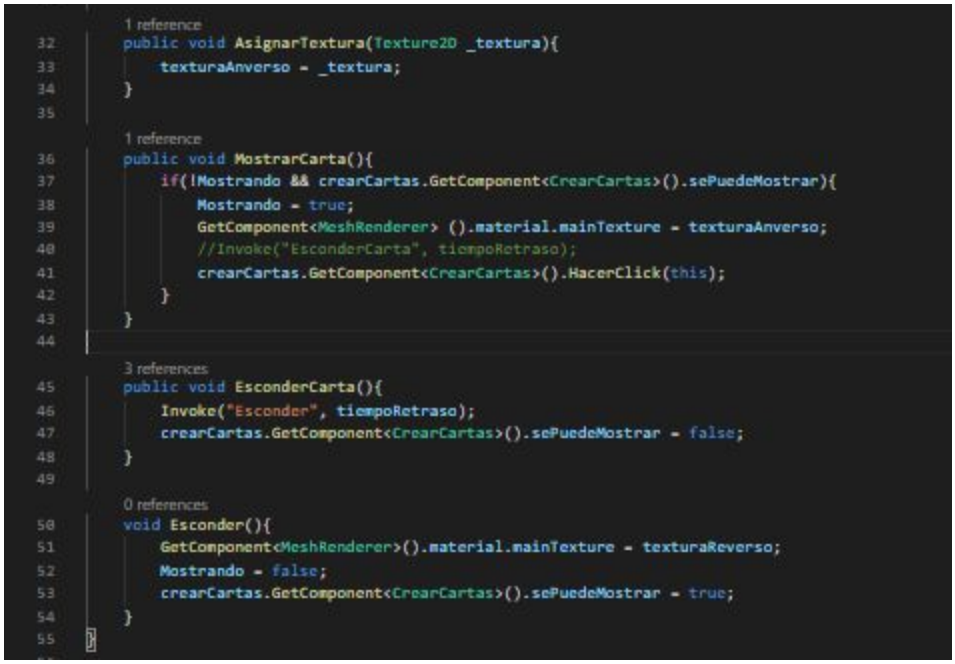
[Fondo del juego]

4.3 Scripts

4.3.1 Script: Carta.cs

Tiene varios métodos pero sirve principalmente para asignar las cartas, para mostrar las cartas y para hacer los clicks cuando le pica a una carta.





```
1 reference
public void AsignarTextura(Texture2D _textura){
    texturaAnverso = _textura;
}

1 reference
public void MostrarCarta(){
    if(!Mostrando && crearCartas.GetComponent<CrearCartas>().sePuedeMostrar){
        Mostrando = true;
        GetComponent<MeshRenderer> ().material.mainTexture = texturaAnverso;
        //Invoke("EsconderCarta", tiempoRetraso);
        crearCartas.GetComponent<CrearCartas>().HacerClick(this);
    }
}

3 references
public void EsconderCarta(){
    Invoke("Esconder", tiempoRetraso);
    crearCartas.GetComponent<CrearCartas>().sePuedeMostrar = false;
}

0 references
void Esconder(){
    GetComponent<MeshRenderer>().material.mainTexture = texturaReverso;
    Mostrando = false;
    crearCartas.GetComponent<CrearCartas>().sePuedeMostrar = true;
}
```

4.3.2 Script: CrearCarta.cs

Este script es el que hace casi todo el trabajo, sirve para poner contador de las parejas encontradas, el contador de los clicks, el que hace que baraje las cartas y siempre sean aleatorios, el que hace que compare dos cartas se comparen a ver si son iguales, el que reinicia las cartas y muestra el contador de los intentos.

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

4 references
public class CrearCartas : MonoBehaviour {

    1 reference
    public GameObject CartaPrefab;
    7 references
    public int ancho;
    1 reference
    public Transform CartasParent;
    15 references
    private List<GameObject> cartas = new List<GameObject> ();
    0 references
    public Material[] materiales;
    3 references
    public Texture2D[] texturas;

    3 references
    public int contadorClicks;
    2 references
    public Text textoContadorIntentos;

    6 references
    public Carta CartaMostrada;
    4 references
    public bool sePuedeMostrar = true;

    2 references
    public InterfazUsuario interfazUsuario;

    3 references
    public int contadorParejasEncontradas;

```

```
0 references
public void Reiniciar(){
    ancho = 0;
    cartas.Clear();
    GameObject[] cartasEliminar = GameObject.FindGameObjectsWithTag("Carta");
    for(int i = 0; i < cartasEliminar.Length; i++){
        Destroy(cartasEliminar[i]);
    }
    contadorClicks = 0;
    textoContadorIntentos.text = "Intentos";
    CartaMostrada = null;
    sePuedeMostrar = true;
    contadorParejasEncontradas = 0;

    Crear();
}

1 reference
public void Crear(){
    ancho = interfazUsuario.dificultad;

    int cont = 0;
    for(int i = 0; i < ancho; i++){
        for(int x = 0; x < ancho; x++){
            float factor = 9.0f / ancho;
            Vector3 posicionTemp = new Vector3(x*factor, 0, i*factor);

            GameObject cartaTemp = Instantiate(CartaPrefab, posicionTemp, Quaternion.Euler(new Vector3(0, 180, 0)));

            cartaTemp.transform.localScale *= factor;

            cartas.Add(cartaTemp);

            cartaTemp.GetComponent<Carta> ().posicionOriginal = posicionTemp;
            //cartaTemp.GetComponent<Carta> ().IdCarta = cont;

            cartaTemp.transform.parent = CartasParent;

            cont++;
        }
    }
    AsignarTexturas();
    Barajar();
}
1 reference
void AsignarTexturas(){
    int[] arrayTemp = new int[texturas.Length];

    for (int i = 0; i <= texturas.Length - 1; i++){
        arrayTemp[i] = i;
    }

    for (int t = 0; t < arrayTemp.Length; t++){
        int tmp = arrayTemp[t];
        int r = Random.Range(t, arrayTemp.Length);
        arrayTemp[t] = arrayTemp[r];
        arrayTemp[r] = tmp;
    }

    int[] arrayDefinitivo = new int[ancho * ancho];

    for (int i = 0; i < arrayDefinitivo.Length; i++){
        arrayDefinitivo[i] = arrayTemp[i];
    }

    for(int i = 0; i < arrayDefinitivo.Length; i++){
        cartas[i].GetComponent<Carta> ().AsignarTextura (texturas[(arrayDefinitivo[i / 2])]);
        cartas[i].GetComponent<Carta> ().IdCarta = i / 2;
    }
}
```

```
    1 reference
    void Barajar(){
        int aleatorio;

        for (int i = 0; i < cartas.Count; i++){
            aleatorio = Random.Range(i, cartas.Count);
            cartas[i].transform.position = cartas [aleatorio].transform.position;
            cartas[aleatorio].transform.position = cartas[i].GetComponent<Carta>().posicionOriginal;

            cartas[i].GetComponent<Carta>().posicionOriginal = cartas[i].transform.position;
            cartas[aleatorio].GetComponent<Carta>().posicionOriginal = cartas[aleatorio].transform.position;
        }
    }
    1 reference
    public void HacerClick(Carta _carta){
        if(CartaMostrada == null){
            CartaMostrada = _carta;
        }else{
            contadorClicks++;
            ActualizarUI();
            if(CompararCartas(_carta.gameObject, CartaMostrada.gameObject)){
                print("¡Enhorabuena! Has encontrado una pareja");
                contadorParejasEncontradas++;
                if (contadorParejasEncontradas == cartas.Count / 2){
                    print("¡Enhorabuena! Has encontrado todas las parejas");
                    interfazUsuario.MostrarMenuGanador();
                }
            }else{
                _carta.EsconderCarta();
                CartaMostrada.EsconderCarta();

            }
            CartaMostrada = null;
        }
    }

    1 reference
    public bool CompararCartas(GameObject carta1, GameObject carta2){
        bool resultado;
        //if(carta1.GetComponent<MeshRenderer> ().material.mainTexture == carta2.GetComponent<MeshRenderer> ().material.mainTexture){
        if(carta1.GetComponent<Carta> ().IdCarta == carta2.GetComponent<Carta> ().IdCarta){
            resultado = true;
        }else{
            resultado = false;
        }return resultado;

    }

    1 reference
    public void ActualizarUI(){
        textoContadorIntentos.text = "Intentos: " + contadorClicks;
    }
}
```

4.3.3 Script: InterfazUsuario.cs

Este script hace los cambios del interfaz de usuario del juego. Hace las acciones

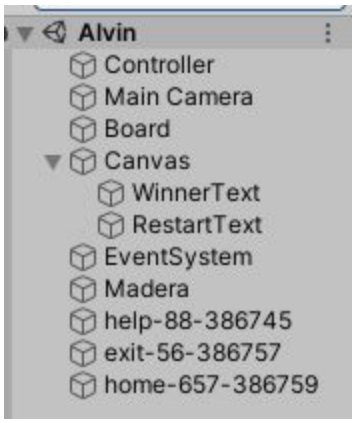
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
2 references
public class InterfazUsuario : MonoBehaviour{
    2 references
    public GameObject menu;
    2 references
    public GameObject menuGanador;
    1 reference
    public Slider sliderDificultad;
    1 reference
    public Text textoDificultad;

    3 references
    public bool menuMostrado;
    2 references
    public bool menuMostradoGanador;
    3 references
    public int dificultad;

    0 references
    void Start() {
        CambiarDificultad();
    }

    0 references
    public void MostrarMenu(){
        menu.SetActive(true);
        menuMostrado = true;
    }

    0 references
    public void EsconderMenu(){
        menu.SetActive(false);
        menuMostrado = false;
    }

    1 reference
    public void MostrarMenuGanador(){
        menuGanador.SetActive(true);
        menuMostradoGanador = true;
    }

    0 references
    public void EsconderMenuGanador(){
        menuGanador.SetActive(false);
        menuMostradoGanador = false;
    }

    1 reference
    public void CambiarDificultad(){
        dificultad = (int)sliderDificultad.value * 2;
        textoDificultad.text = "Dificultad: " + dificultad;
    }
}
```

5. Ajedrez

El juego que se estuvo trabajando durante el semestre fue un ajedrez. El juego fue hecho en Unity, con diversas herramientas, como los sprites, scripts, o las herramientas que ofrece unity.

El juego tiene como objetivo que los jugadores puedan desarrollar su capacidad de pensar, así como la de tomar decisiones y riesgos para lograr sus metas. Un juego en donde en un movimiento ya sea, lo puedes tener todo, o lo puedes perder.

5.1 Componentes de la escena del juego

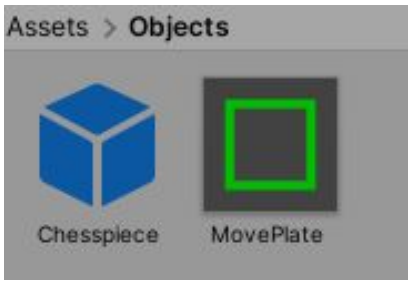


5.2 Sprites y prefabs

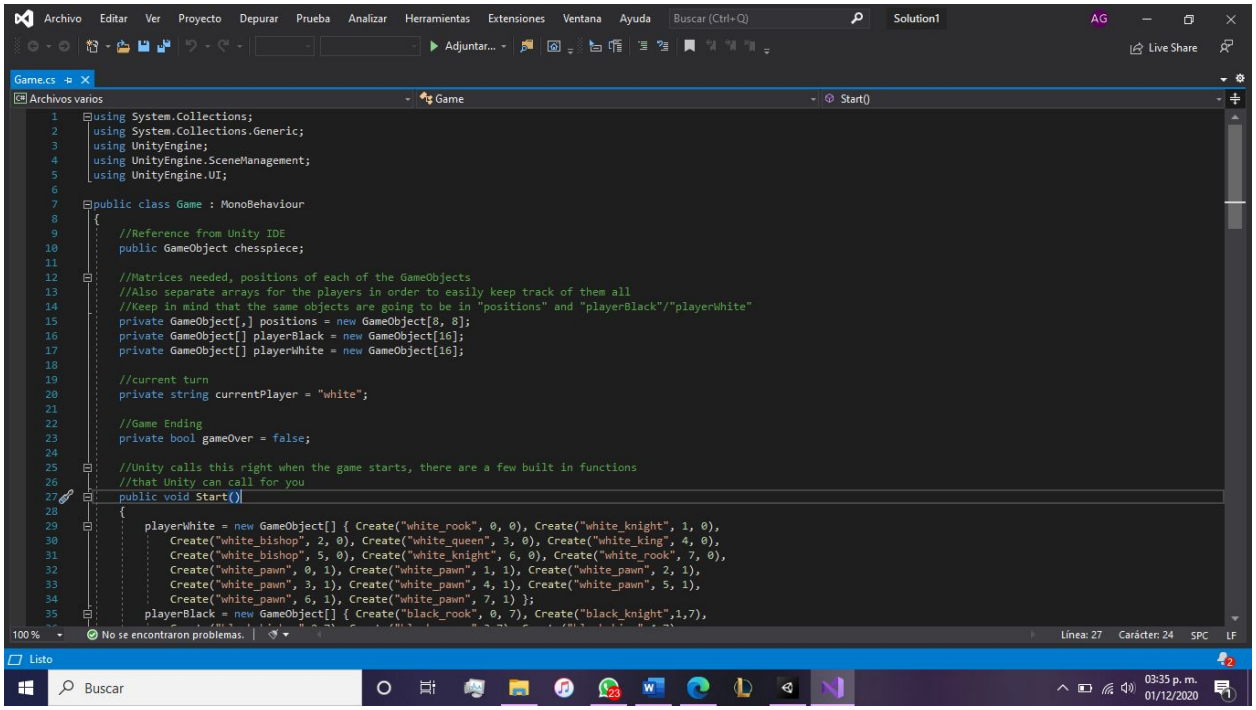
Para el juego se han utilizado dos archivos como base:

- El tablero del juego fue diseñado en Paint, y fue importado.
- La piezas del juego fueron descargadas de internet, posteriormente fueron recortadas cada una, de tal manera que puedan usarse las piezas individualmente

Prefabs:



5.3 Scripts



6. TicTacToe

6.1 Tutoriales usados

https://youtu.be/FBamzpJTlG4?list=PLkzh1bySTmYB83ybePBUtsP4t0DAdspiw

6.2 Sprites

6.3 Scripts

Se usaron dos scripts: Game Controller y GridSpace.

6.3.1 Script: GameController.cs

En el Game Controller están todas las clase que hacen con que el juego funcione, y es la responsable por todos los movimientos que se hacen dentro del juego, como por ejemplo la clase Player para los jugadores desde las imágenes y los botones.

```
 6      [System.Serializable]
 7    public class Player
 8    {
 9        public Image panel;
10        public Text text;
11        public Button button;
12    }
13
14      [System.Serializable]
15    public class PlayerColor
16    {
17        public Color panelColor;
18        public Color textColor;
19    }
```

```
21  public class GameController : MonoBehaviour
22  {
23
24      public Text[] buttonList;
25      public GameObject gameOverPanel;
26      public Text gameOverText;
27      public GameObject restartButton;
28
29      public Player playerX;
30      public Player playerO;
31      public PlayerColor activePlayerColor;
32      public PlayerColor inactivePlayerColor;
33
34      public GameObject startInfo;
35      public bool playerMove;
36      public float delay;
37
38      private string playerSide;
39      private int moveCount;
40      private string computerSide;
41      private int value;
```

```
43      void Awake()
44      {
45          SetGameControllerReferenceOnButtons();
46          gameOverPanel.SetActive(false);
47          moveCount = 0;
48          restartButton.SetActive(false);
49          playerMove = true;
50
51      }
52
53      void Update()
54      {
55          if(playerMove == false)
56          {
57              delay += delay * Time.deltaTime;
58              if(delay >= 100)
59              {
60                  value = Random.Range(0, 8);
61                  if(buttonList [value].GetComponentInParent<Button> ().interactable == true)
62                  {
63                      buttonList[value].text = GetComputerSide();
64                      buttonList[value].GetComponentInParent<Button>().interactable = false;
65                      EndTurn();
66                  }
67              }
68          }
69      }
```

```
71      void SetGameControllerReferenceOnButtons()
72      {
73          for (int i = 0; i < buttonList.Length; i++)
74          {
75              buttonList[i].GetComponentInParent<GridSpace>().SetGameControllerReference(this);
76          }
77      }
78
79      public void SetStartingSide(string startingSide)
80      {
81          playerSide = startingSide;
82          if(playerSide == "X")
83          {
84              computerSide = "O";
85              SetPlayerColors(playerX, playerO);
86          }
87          else
88          {
89              computerSide = "X";
90              SetPlayerColors(playerO, playerX);
91
92          }
93          StartGame();
94      }
```

```
96      void StartGame()
97      {
98          SetBoardInteractable(true);
99          SetPlayerButtons(false);
100         startInfo.SetActive(false);
101     }
102
103     public string GetPlayerSide()
104     {
105         return playerSide;
106     }
107
108     public string GetComputerSide()
109     {
110         return computerSide;
111     }
```

```
113     public void EndTurn()
114     {
115         moveCount++;
116
117         if (buttonList[0].text == playerSide && buttonList[1].text == playerSide && buttonList[2].text == playerSide)
118         {
119             GameOver(playerSide);
120         }
121
122         else if (buttonList[3].text == playerSide && buttonList[4].text == playerSide && buttonList[5].text == playerSide)
123         {
124             GameOver(playerSide);
125         }
126
127         else if (buttonList[6].text == playerSide && buttonList[7].text == playerSide && buttonList[8].text == playerSide)
128         {
129             GameOver(playerSide);
130         }
131
132         else if (buttonList[0].text == playerSide && buttonList[3].text == playerSide && buttonList[6].text == playerSide)
133         {
134             GameOver(playerSide);
135         }
136
137         else if (buttonList[1].text == playerSide && buttonList[4].text == playerSide && buttonList[7].text == playerSide)
```

```
210     void SetPlayerColors(Player newPlayer, Player oldPlayer)
211     {
212         newPlayer.panel.color = activePlayerColor.panelColor;
213         newPlayer.text.color = activePlayerColor.textColor;
214         oldPlayer.panel.color = activePlayerColor.panelColor;
215         oldPlayer.text.color = activePlayerColor.textColor;
216     }
217
218     void GameOver(string winningPlayer)
219     {
220         SetBoardInteractable(false);
221
222         if (winningPlayer == "empate")
223         {
224             SetGameOverText("Empate");
225         }
226         else
227         {
228             SetGameOverText(winningPlayer + " Gana");
229         }
230
231         restartButton.SetActive(true);
232     }
```

```
256     public void RestarGame()
257     {
258
259         moveCount = 0;
260         gameOverPanel.SetActive(false);
261         restartButton.SetActive(false);
262         SetPlayerButtons(true);
263         SetPlayerColorsInactive();
264         startInfo.SetActive(true);
265         playerMove = true;
266         delay = 10;
267
268
269         for (int i = 0; i < buttonList.Length; i++)
270         {
271             buttonList[i].text = "";
272         }
273         SetPlayerColors(playerX, playerO);
274         restartButton.SetActive(false);
275     }
276
277     void SetBoardInteractable(bool toggle)
278     {
279         for (int i = 0; i < buttonList.Length; i++)
280         {
281             buttonList[i].GetComponentInParent<Button>().interactable = toggle;
```

```
285     void SetPlayerButtons(bool toggle)
286     {
287         playerX.button.interactable = toggle;
288         playerO.button.interactable = toggle;
289     }
290
291     void SetPlayerColorsInactive()
292     {
293         playerX.panel.color = inactivePlayerColor.panelColor;
294         playerX.text.color = inactivePlayerColor.textColor;
295         playerO.panel.color = inactivePlayerColor.panelColor;
296         playerO.text.color = inactivePlayerColor.textColor;
297     }
298 }
```

6.3.2 Script: GridSpace.cs

```
using UnityEngine;
using UnityEngine.UI;


public class GridSpace : MonoBehaviour
{
    public Button button;
    public Text buttonText;

    private GameController gameController;

    public void SetGameControllerReference(GameController controller)
    {
        gameController = controller;
    }

    public void SetSpace()
    {
        if(gameController.playerMove == true)
        {
            buttonText.text = gameController.GetPlayerSide();
            button.interactable = false;
            gameController.EndTurn();
        }
    }
}

```

Manual de usuario

1. Introducción

Este juego consiste en una serie de juegos que ayudan a los conquistadores a completar, entender y practicar especialidades. Son aproximadamente 5 especialidades, y se pueden agregar muchísimas más. Se van acumulando puntos en cada juego y el objetivo final es completar las especialidades con el mayor puntaje posible.

1.1 Descripción

Composiciòn de la escena: Primeramente se compone de un menú general para seleccionar la especialidad a practicar, la configuración del juego y el puntaje. En futuros trabajos estos puntajes se pueden actualizar vía internet para poder competir contra otros jugadores.
Género: cualquier público, principalmente jóvenes conquistadores o maestros que trabajan en el área de conquistadores.
Consiste en los siguientes juegos:

- Especialidad de nudos
- Memorama de personajes bíblicos
- Tic-tac-toe
- Damas inglesas
- Ajedrez

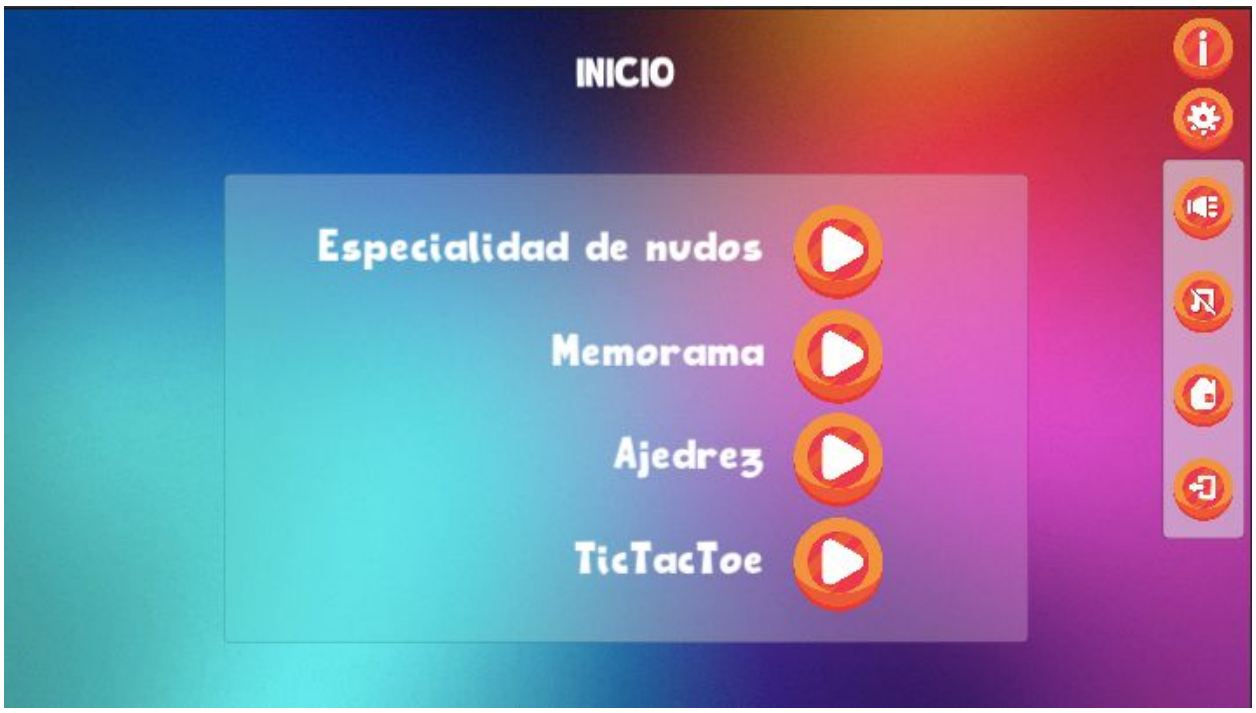
1.2 Requisitos mínimos del sistema

Plataforma: PC
SO: Windows 7/8/10
Procesador: Intel Pentium Dual Core / AMD FX
Memoria: 2 GB RAM
Tarjeta de Video: Intel HD Graphics

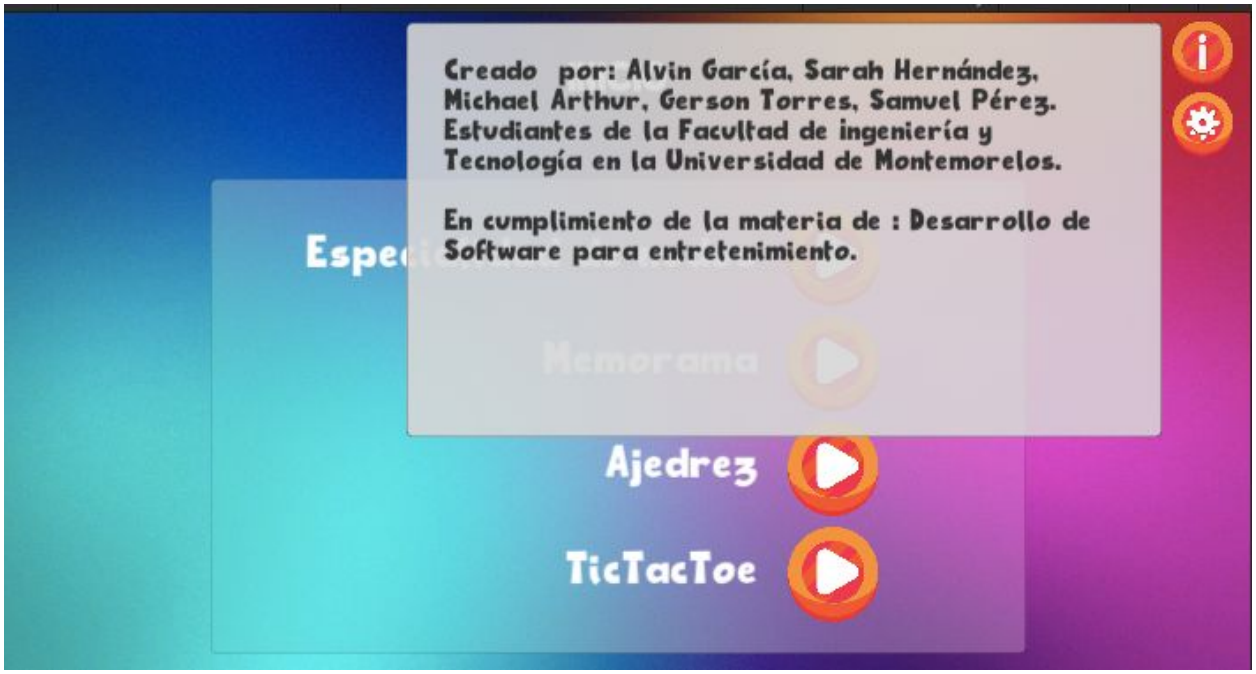
Almacenamiento: 100 MB de espacio libre
Tarjeta de Sonido: DirectX

2. Pantalla principal

Como en todas las escenas, la pantalla principal contiene un menú de opciones donde el usuario puede seleccionar si mutear o encender el sonido de click durante el jueg, si mutear o encender la música, si regresar a la pantalla principal o si desea salir del juego:



Además como todas las escenas contiene una descripción general del todo el juego:



3. Test de especialidad de nudos

3.1 Número de jugadores del juego

Un jugador.

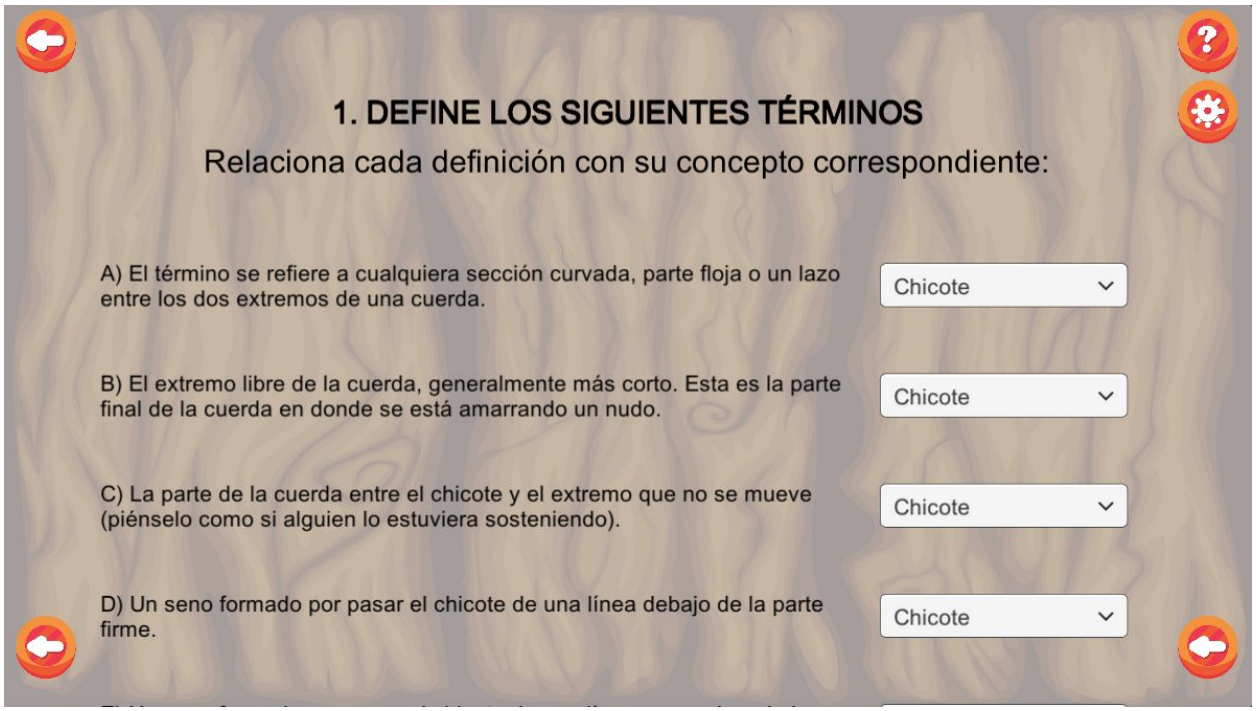
3.2 Cómo se juega

1. Primeramente el usuario selecciona el botón de comenzar juego y posteriormente se carga la pantalla de la primera pregunta.
2. El usuario debe ir seleccionando las respuestas correctas a cada pregunta para completar la especialidad. Puede dar scroll para ver las preguntas completas.
3. Al llegar a la pregunta final es necesario el programa preguntará si desea terminar. Si ya está seguro de haber terminado puede seleccionar el botón de sí, de lo contrario selecciona el botón de no y puede corregir las preguntas en caso de tener duda con sus respuestas dadas.
4. Al terminar el juego puede completar y evaluar su especialidad.

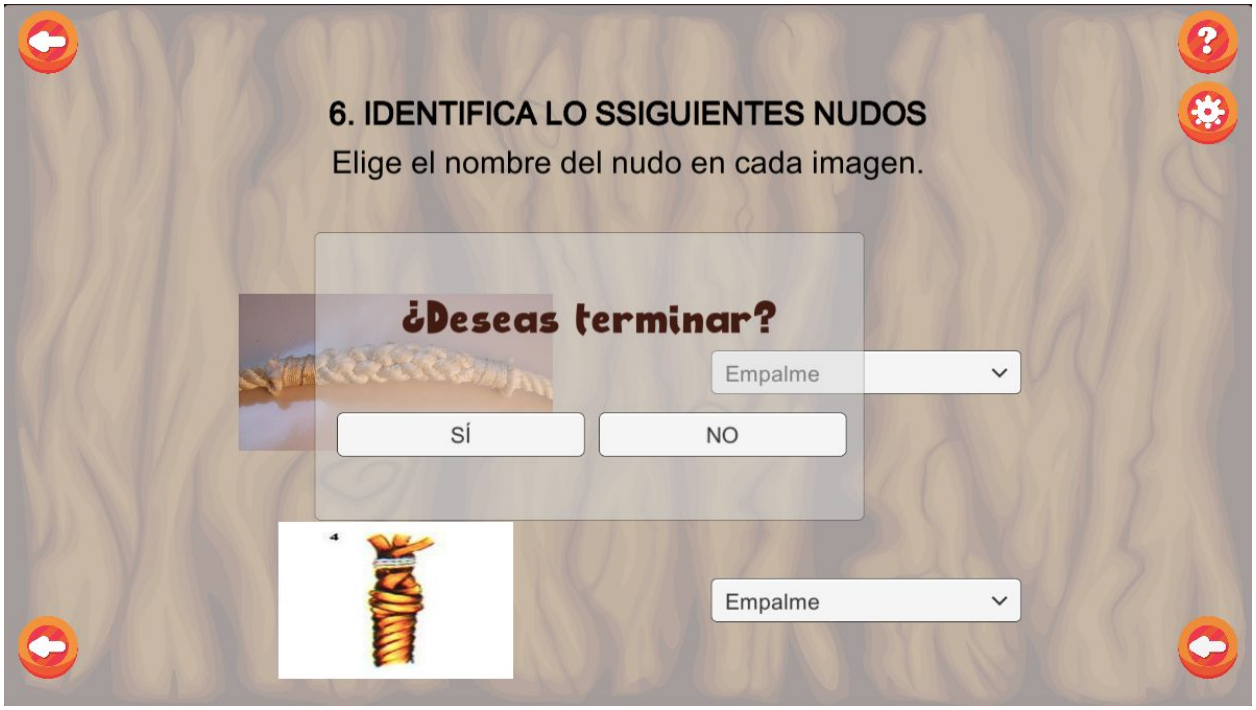
3.3 Capturas de pantalla

Comenzar test:

Contestar según se indica:



Panel de finalizar juego:



4. Memorama

En este juego el objetivo es que el jugador encuentre todos los pares de cartas.

4.1 Número de jugadores del juego

Un jugador.

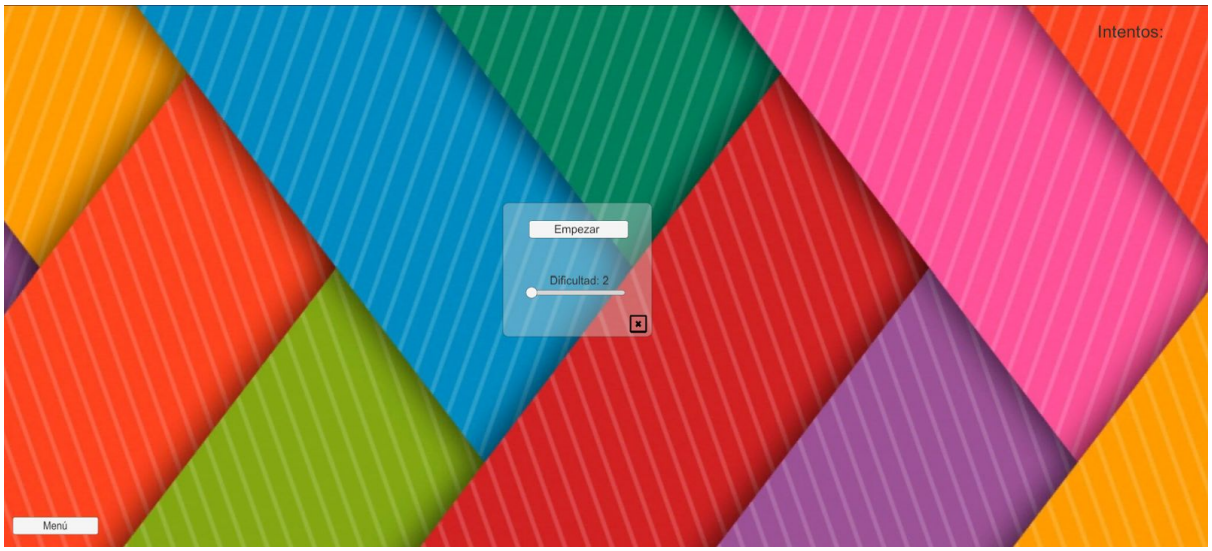
4.2 Cómo se juega

A continuación los pasos para jugar el memorama de personajes bíblicos.

1. Seleccionar la dificultad con la barra cursora.
2. Darle a empezar
3. Seleccionar cartas y encontrar parejas, cada dos cartas que se seleccione se vuelve a tapar y el chiste es encontrar todas las cartas.
4. Empezar si gustas y seleccionar la dificultad.
5. Ganas cuando has encontrado la pareja de todas las cartas

4.3 Capturas de pantalla

Pantalla de seleccionar dificultad:



Pantalla de juego:

Pantalla de juego terminado donde puedes saber si perdiste o ganaste:

5. Ajedrez

El juego está compartido con otros más, por lo cual hay una opción para salir del juego y poder ir a otro, o simplemente cerrar la aplicación.

El juego tiene como objetivo que los jugadores puedan desarrollar su capacidad de pensar, así como la de tomar decisiones y riesgos para lograr sus metas. Un juego en donde en un movimiento ya sea, lo puedes tener todo, o lo puedes perder.

5.1 Número de jugadores del juego

Este juego esta diseñado para jugador vs jugador, en donde se buscará derrotar al rey contrario para obtener la victoria.

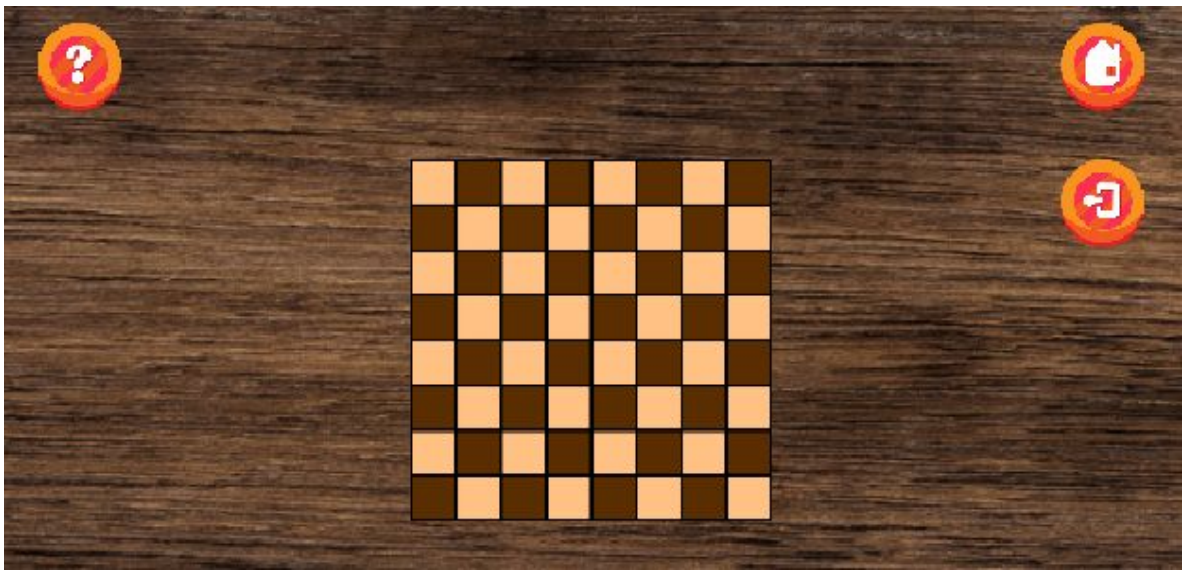
5.2 Cómo se juega

En este juego de estrategia se desafía a otro jugador. Al seleccionar una pieza te mostrará las posibles posiciones en donde se podrá mover la pieza, en caso de que pueda comer otra pieza esta se mostraba de otro color. Este es un juego en el que se terminará cuando se coma al rey en jaque mate.

Cada jugador moverá una pieza por turno. Durante el juego, te muestra las posibles localizaciones en donde se pueda mover en color verde, y en caso de que puedas comer una pieza, se mostrara de color rojo.

5.3 Capturas de pantalla

Tablero de inicio



Juego en proceso

6. TicTacToe

6.1 Número de jugadores del juego

Dos jugadores.

6.2 Cómo se juega

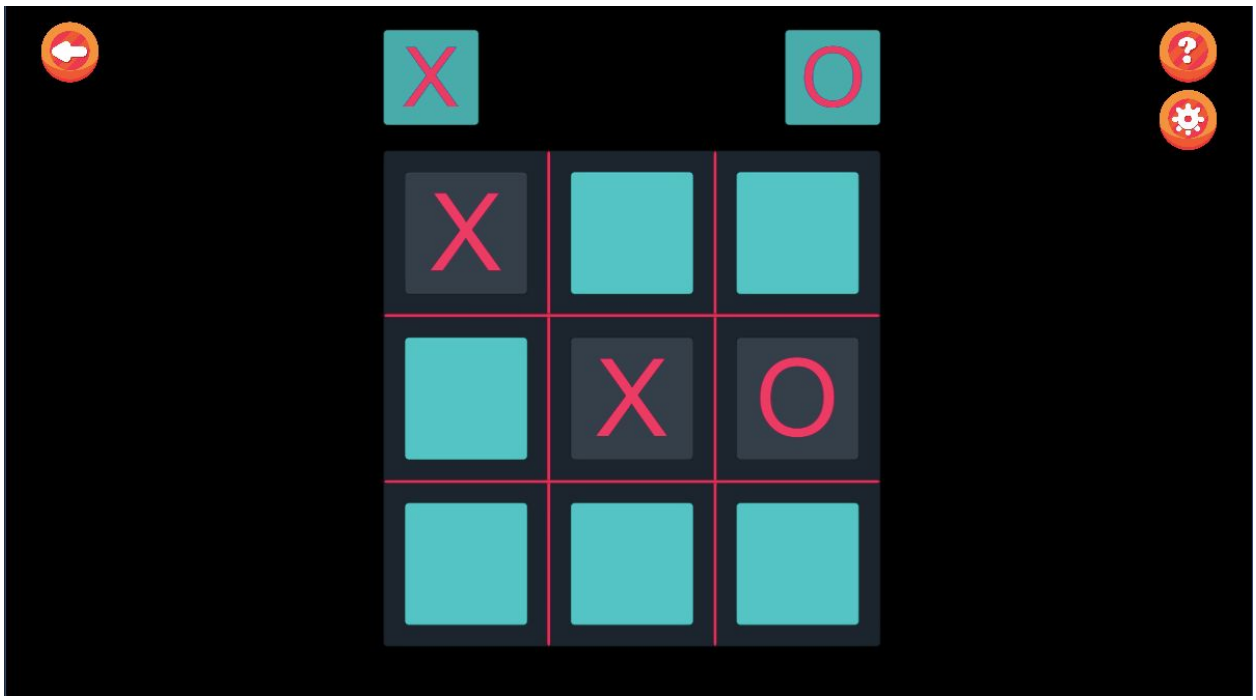
¿Cómo se juega? (pasos para jugarlo):

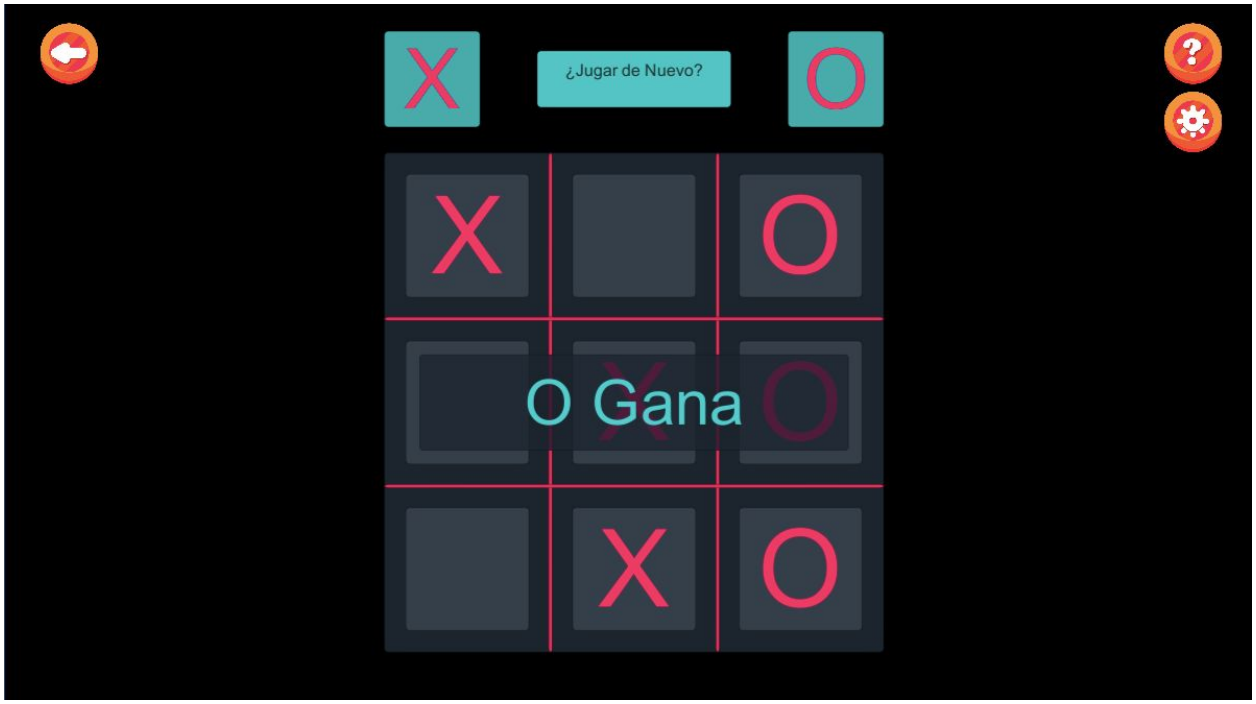
Se juega de forma sencilla, con dos personas o de forma automática (o con la compu)

¿Cómo ganar?

Para ganar los competidores tienen que alinear “3X” o “3O” de forma horizontal, vertical o diagonal en la izquierda o derecha.

6.3 Capturas de pantalla




X
¿Jugar de Nuevo?
O
X
O
O Gana
X
O