TRABAJO FIN DE GRADO

Manual de código

Grado en Ingeniería Informática

Stealthy Spy, un juego de plataformas para dispositivos Android

Autor: Alberto Adamuz Priego

Directores: Juan Carlos Fernández Caballero

David Guijo Rubio

6 de junio de 2023

UNIVERSIDAD D CORDOBA

Firma de conformidad del autor y del director del trabajo de fin de grado:

- **Autor:**

Fdo: Alberto Adamuz Priego

- **Directores:**

Fdo: Juan Carlos Fernández Caballero

Fdo: David Guijo Rubio

Índice de contenidos

CAPÍTULO 1

INTRODUCCIÓN

Este manual ha sido elaborado con el fin de proporcionar una visión detallada del código fuente de la aplicación desarrollada en este Trabajo Fin de Grado. Su objetivo principal es brindar a los desarrolladores, colaboradores y otros interesados una comprensión completa de la estructura, funcionalidad y diseño interno del software.

En este proyecto se ha utilizado el motor gráfico *Unity* el cual funciona en base a *scripts*. Estos se utilizan como componentes que otorgan cierta funcionalidad a los objetos que los poseen. Aquí se va a abordar los *scripts* creados para esta aplicación sin entrar en los *scripts* propios del motor.

El código del proyecto se encuentra disponible en el siguiente repositorio https://github.com/AlbertoAdamuzPriego/StealthySpy

Los *scripts* se almacenan en la carpeta *Assets/Scripts/* y estos se han dividido en dos subcarpetas o paquetes:

- ***Menu.*** Contiene los *scripts* que se encargan de controlar la aplicación y los menús principales.

- ***Gameplay.*** Son aquellos que únicamente se utilizan durante el desarrollo de una partida.

CAPÍTULO 2

PAQUETE MENU

Este capítulo expone el código fuente de los *scripts* del paquete *Menu*. Este paquete recoge los *scripts* encargados del manejo de los principales módulos de la aplicación.

A continuación se expondrá cada uno:

2.1. GameManager

Este *script* representa la clase principal de la aplicación. Esta controla los estados de la aplicación a través de eventos. Cuando se produce un cambio de estado se llama a uno de estos eventos y este invoca a todas funciones de todos los *scripts* suscritas a él.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System;
5  using TMPro;
6  using JetBrains.Annotations;
7
8  //Es la clase principal que se encarga de gestionar la aplicacion
9  public class GameManager : MonoBehaviour
10 {
11     /*
12     * Estos eventos llaman a una serie de funciones suscritas a
           ellos cuando son invocados
13     * permitiendo la comunicacion y el control instantaneo con
           otros scripts
14     */
15     public event Action OnMainMenu; //Estado en menu principal
16     public event Action OnSettingsMenu; //Estado en menu de
           ajustes
17     public event Action OnLevelMenu; //Estado en menu de niveles
18     public event Action OnCreditsMenu; //Estado en menu de
           creditos
19     public event Action OnScoreMenu; //Estado en menu de
           puntuaciones
```

```
20      public event Action ChangeDifficulty; //Evento al cambiar la
            dificultad
21
22      public static GameManager instance; //Instancia del script (
            Patron Singleton)
23
24      public int finishMode; //Variable que sirve para identificar
            como ha finalizado una partida
25
26      [SerializeField] private TMP_Text difficultyLabel; // Texto
            del selector de dificultad
27
28      //Controla que solo exista una instancia de GameManager
29      private void Awake()
30      {
31          if(instance!=null && instance!=this)
32          {
33              Destroy(gameObject);
34          }
35
36          else
37          {
38              instance = this;
39          }
40      }
41
42      void Start()
43      {
44          LoadData();
45      }
46
```

```
47      //Activa el menu principal
48      public void MainMenu()
49      {
50          //Ejecuta las funciones suscritas al evento OnMainMenu
51          OnMainMenu?.Invoke();
52
53      }
54
55      //Activa el menu de ajustes
56      public void SettingsMenu()
57      {
58          //Ejecuta las funciones suscritas al evento
                OnSettingsMenu
59          OnSettingsMenu?.Invoke();
60
61      }
62
63      //Activa el menu de niveles
64      public void LevelMenu()
65      {
66          //Ejecuta las funciones suscritas al evento OnLevelMenu
67
68          OnLevelMenu?.Invoke();
69
70          //PlayerPrefs.SetInt("finish", -1);
71
72      }
73
74      //Activa el menu de creditos
75      public void CreditsMenu()
76      {
```

```
77          //Ejecuta las funciones suscritas al evento OnLevelMenu
78          OnCreditsMenu?.Invoke();
79
80      }
81
82      //Activa el menu de puntuaciones
83      public void ScoreMenu()
84      {
85          //Ejecuta las funciones suscritas al evento OnScoreMenu
86          OnScoreMenu?.Invoke();
87
88      }
89
90      //Cierra el juego
91      public void CloseGame()
92      {
93          PlayerPrefs.DeleteKey("finish");
94         // PlayerPrefs.DeleteAll();
95          Application.Quit();
96      }
97
98      //Activa el evento de cambio de dificultad
99      public void Difficulty()
100     {
101         ChangeDifficulty?.Invoke();
102     }
103
104     //Carga los datos necesarios para la aplicacion
105     private void LoadData()
106     {
107         //Carga la dificultad
```

```
108         int difficulty = PlayerPrefs.GetInt("difficulty");
109         switch (difficulty)
110         {
111             case 0: difficultyLabel.SetText("Facil"); break;
112 
113             case 1: difficultyLabel.SetText("Normal"); break;
114 
115             case 2: difficultyLabel.SetText("Dificil"); break;
116         }
117 
118         //Carga el modo de finalizacion para la partida (-1 = no
                hay partida)
119         finishMode = PlayerPrefs.GetInt("finish",-1);
120 
121         //EL jugador supero la partida
122         if(finishMode==0 )
123         {
124             //Se actualiza primero las puntuaciones para evitar
                    bugs
125             FindAnyObjectByType<MapManager>().RecalculateScores()
                    ;
126             LevelMenu();
127         }
128 
129         //El jugador perdio la partida o la abandono
130         else
131         {
132 
133             MainMenu();
134         }
135 
```

```
136             //Se actualiza a -1 de nuevo
137             PlayerPrefs.SetInt("finish", -1);
138         }
139 
140 }
```

2.2. UIManager

Este *script* controla la interfaz del juego encargándose principalmente de la transición entre los menús.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using DG.Tweening;
5  using System;
6  using UnityEngine.UI;
7
8  public class UIManager : MonoBehaviour
9  {
10     //Canvas de cada menu
11     [SerializeField] private GameObject mainMenuCanvas;
12     [SerializeField] private GameObject settingsMenuCanvas;
13     [SerializeField] private GameObject mapMenuCanvas;
14     [SerializeField] private GameObject scoreMenuCanvas;
15     [SerializeField] private GameObject creditsMenuCanvas;
16     [SerializeField] private GameObject loadingCanvas;
17
18     void Start()
19     {
20         //Suscribimos cada funcion a su evento correspondiente
21         GameManager.instance.OnMainMenu += activateMainMenu;
22
23         GameManager.instance.OnSettingsMenu +=
               activateSettingsMenu;
24
```

```
25          GameManager.instance.OnLevelMenu += activateMapMenu;
26
27          GameManager.instance.OnScoreMenu += activateScoreMenu;
28
29          GameManager.instance.OnCreditsMenu += activateCreditsMenu
                ;
30      }
31
32      //Activa el menu principal
33      private void activateMainMenu()
34      {
35          //Escala los objetos a una determinada escala en una
                cantidad de segundos (0.3)
36          mainMenuCanvas.transform.GetChild(0).transform.DOScale(
                new Vector3(1, 1, 1), 0.3f);
37          mainMenuCanvas.transform.GetChild(1).transform.DOScale(
                new Vector3(1, 1, 1), 0.3f);
38          mainMenuCanvas.transform.GetChild(2).transform.DOScale(
                new Vector3(1.5f, 1.5f, 1), 0.3f);
39          mainMenuCanvas.transform.GetChild(3).transform.DOScale(
                new Vector3(1.5f, 1.5f, 1), 0.3f);
40          mainMenuCanvas.transform.GetChild(4).transform.DOScale(
                new Vector3(1.5f, 1.5f, 1.5f), 0.3f);
41          mainMenuCanvas.transform.GetChild(5).transform.DOScale(
                new Vector3(0.5f, 0.5f, 1), 0.3f);
42          mainMenuCanvas.transform.GetChild(6).transform.DOScale(
                new Vector3(0.5f, 0.5f, 0), 0.3f);
43
44          //Mueve el objeto en el eje Y
45          settingsMenuCanvas.transform.DOMoveY(2000, 0.3f);
46
```

```
47          mapMenuCanvas.transform.GetChild(0).transform.DOScale(new
                Vector3(0, 0, 0), 0.3f);
48          mapMenuCanvas.transform.GetChild(1).transform.DOScale(new
                Vector3(0, 0, 0), 0.3f);
49          mapMenuCanvas.transform.GetChild(2).transform.DOScale(new
                Vector3(0, 0, 0), 0.3f);
50          mapMenuCanvas.transform.GetChild(3).transform.DOScale(new
                Vector3(0, 0, 0), 0.3f);
51          mapMenuCanvas.transform.GetChild(4).transform.DOScale(new
                Vector3(0, 0, 1), 0.3f);
52          mapMenuCanvas.transform.GetChild(5).transform.DOScale(new
                Vector3(0, 0, 1), 0.3f);
53
54          scoreMenuCanvas.transform.GetChild(0).transform.DOScale(
               new Vector3(0, 0, 0), 0f);
55          scoreMenuCanvas.transform.GetChild(1).transform.DOScale(
               new Vector3(0, 0, 0), 0f);
56          scoreMenuCanvas.transform.GetChild(2).transform.DOScale(
               new Vector3(0, 0,0), 0f);
57          scoreMenuCanvas.transform.GetChild(3).transform.DOScale(
               new Vector3(0, 0, 0), 0f);
58          scoreMenuCanvas.transform.GetChild(4).transform.DOScale(
               new Vector3(0, 0, 0), 0f);
59          scoreMenuCanvas.transform.GetChild(5).transform.DOScale(
               new Vector3(0, 0, 0), 0f);
60          scoreMenuCanvas.transform.GetChild(6).transform.DOScale(
               new Vector3(0, 0, 0), 0f);
61
62          creditsMenuCanvas.transform.GetChild(0).transform.DOScale
               (new Vector3(0, 0, 0), 0.3f);
63          creditsMenuCanvas.transform.GetChild(1).transform.DOScale
```

```
            (new Vector3(0, 0, 0), 0.3f);
64          creditsMenuCanvas.transform.GetChild(2).transform.DOScale
            (new Vector3(0, 0, 0), 0.3f);
65          creditsMenuCanvas.transform.GetChild(3).transform.DOScale
            (new Vector3(0, 0, 0), 0.3f);
66
67
68      }
69
70      //Activa el menu de configuracion
71      private void activateSettingsMenu()
72      {
73
74          settingsMenuCanvas.transform.DOMoveY(510, 0.3f);
75
76      }
77
78      //Activa el menu de selector de mapa
79      private void activateMapMenu()
80      {
81              mainMenuCanvas.transform.GetChild(0).transform.
                DOScale(new Vector3(0, 0, 0), 0.3f);
82              mainMenuCanvas.transform.GetChild(1).transform.
                DOScale(new Vector3(0, 0, 0), 0.3f);
83              mainMenuCanvas.transform.GetChild(2).transform.
                DOScale(new Vector3(0, 0, 0), 0.3f);
84              mainMenuCanvas.transform.GetChild(3).transform.
                DOScale(new Vector3(0, 0, 0), 0.3f);
85              mainMenuCanvas.transform.GetChild(4).transform.
                DOScale(new Vector3(0, 0, 0), 0.3f);
86              mainMenuCanvas.transform.GetChild(5).transform.
```

```
            DOScale(new Vector3(0, 0, 0), 0.3f);
87          mainMenuCanvas.transform.GetChild(6).transform.
            DOScale(new Vector3(0, 0, 0), 0.3f);
88
89
90          mapMenuCanvas.transform.GetChild(0).transform.DOScale
            (new Vector3(1, 1, 1), 0.3f);
91          mapMenuCanvas.transform.GetChild(1).transform.DOScale
            (new Vector3(1.5f, 1.5f, 1), 0.3f);
92          mapMenuCanvas.transform.GetChild(2).transform.DOScale
            (new Vector3(1, 1, 1), 0.3f);
93          mapMenuCanvas.transform.GetChild(3).transform.DOScale
            (new Vector3(3, 3, 1), 0.3f);
94          mapMenuCanvas.transform.GetChild(4).transform.DOScale
            (new Vector3(-1, 1, 1), 0f);
95          mapMenuCanvas.transform.GetChild(5).transform.DOScale
            (new Vector3(1, 1, 1), 0f);
96
97      }
98
99      //Activa el menu de puntuaciones
100     private void activateScoreMenu()
101     {
102         mainMenuCanvas.transform.GetChild(0).transform.DOScale(
            new Vector3(0, 0, 0), 0.3f);
103         mainMenuCanvas.transform.GetChild(1).transform.DOScale(
            new Vector3(0, 0, 0), 0.3f);
104         mainMenuCanvas.transform.GetChild(2).transform.DOScale(
            new Vector3(0, 0, 0), 0.3f);
105         mainMenuCanvas.transform.GetChild(3).transform.DOScale(
            new Vector3(0, 0, 0), 0.3f);
```

```
106         mainMenuCanvas.transform.GetChild(4).transform.DOScale(
                new Vector3(0, 0, 0), 0.3f);
107         mainMenuCanvas.transform.GetChild(5).transform.DOScale(
                new Vector3(0, 0, 0), 0.3f);
108         mainMenuCanvas.transform.GetChild(6).transform.DOScale(
                new Vector3(0, 0, 0), 0.3f);
109
110         scoreMenuCanvas.transform.GetChild(0).transform.DOScale(
                new Vector3(1, 1, 1), 0.3f);
111         scoreMenuCanvas.transform.GetChild(1).transform.DOScale(
                new Vector3(1, 1, 1), 0.3f);
112         scoreMenuCanvas.transform.GetChild(2).transform.DOScale(
                new Vector3(1, 1, 1), 0.3f);
113         scoreMenuCanvas.transform.GetChild(3).transform.DOScale(
                new Vector3(0.9f, 0.9f, 1), 0.3f);
114         scoreMenuCanvas.transform.GetChild(4).transform.DOScale(
                new Vector3(1.5f, 1.5f, 1), 0.3f);
115         scoreMenuCanvas.transform.GetChild(5).transform.DOScale(
                new Vector3(-1f, 1f, 0), 0.3f);
116         scoreMenuCanvas.transform.GetChild(6).transform.DOScale(
                new Vector3(1f, 1f, 0), 0.3f);
117         scoreMenuCanvas.GetComponentInChildren<
                HorizontalLayoutGroup>().gameObject.transform.DOMoveX
                (0, 0f);
118
119     }
120
121     //Activa el menu de creditos
122     private void activateCreditsMenu()
123     {
124
```

```
125         creditsMenuCanvas.transform.GetChild(0).transform.DOScale
               (new Vector3(1, 1, 0), 0.3f);
126         creditsMenuCanvas.transform.GetChild(1).transform.DOScale
               (new Vector3(1.5f, 1.5f, 0), 0.3f);
127         creditsMenuCanvas.transform.GetChild(2).transform.DOScale
               (new Vector3(1, 1, 0), 0.3f);
128         creditsMenuCanvas.transform.GetChild(3).transform.DOScale
               (new Vector3(1, 1, 0), 0.3f);
129     }
130
131     //Activa la pantalla de carga
132     public void ActivateLoadingCanvas()
133     {
134         loadingCanvas.SetActive(true);
135     }
136 }
```

2.3. Map

Este *script* es un *Scriptable Object*. Esto significa que se pueden crear objetos (no instanciables en la escena) en el editor. Es como una plantilla que puede almacenar la información que se suministra desde el editor. Se utiliza para guardar la información de cada mapa y editarla directamente desde el editor.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using Unity.VisualScripting;
4  using UnityEngine;
5  using UnityEngine.Assertions;
6
7  [CreateAssetMenu] //Pemrite crear objetos en el editor
8
9  //Contiene toda la informacion de un mapa
10 public class Map : ScriptableObject
11 {
12     public string mapName="";
13     public string sceneName; //Escena de Unity con el mapa
14     public float[] score0 = new float[10]; //Puntuaciones
           dificultad facil
15     public float[] score1 = new float[10]; //Puntuaciones
           dificultad normal
16     public float[] score2 = new float[10]; //Puntuaciones
           dificultad dificil
17     public float averageScore0; //Puntuacion media dificultad
           facil
18     public float averageScore1; //Puntuacion media dificultad
           normal
```

```
19      public float averageScore2; //Puntuacion media dificultad
            dificil
20
21      //Devuelve la lista de puntuaciones de una dificultad
22      public float[] GetScoreList(int difficulty)
23      {
24          switch(difficulty)
25          {
26              case 0: return score0;
27              case 1: return score1;
28              case 2: return score2;
29          }
30
31          return score0;
32      }
33
34      //Asigna una lista de puntuaciones a una dificultad
35
36      public void SetScoreList(float[] newScore, int difficulty)
37      {
38          Assert.AreEqual(10, newScore.Length);
39
40          switch(difficulty)
41          {
42              case 0:
43                  score0 = newScore;
44                  break;
45
46              case 1:
47                  score1 = newScore;
48                  break;
```

```
49
50              case 2:
51                  score2 = newScore;
52                  break;
53          }
54
55          CalculateAverages();
56
57      }
58
59      //Devuelve la media de las puntuaciones de una dificultad
60      public float GetAverageScore(int difficulty)
61      {
62          switch(difficulty)
63          {
64              case 0: return averageScore0;
65              case 1: return averageScore1;
66              case 2: return averageScore2;
67          }
68          return averageScore0;
69      }
70
71      //Guarda las puntuaciones diferenciandolas con el nombre del
            mapa
72      public void SaveScore()
73      {
74          for(int i=0;i<score0.Length; i++)
75          {
76
77              PlayerPrefs.SetFloat("Score0" + mapName + i, score0[i
                    ]);
```

```
78              PlayerPrefs.SetFloat("Score1" + mapName + i, score1[i
                    ]);
79              PlayerPrefs.SetFloat("Score2" + mapName + i, score2[i
                    ]);
80          }
81
82          PlayerPrefs.DeleteKey("Average0" + mapName);
83          PlayerPrefs.DeleteKey("Average1" + mapName);
84          PlayerPrefs.DeleteKey("Average2" + mapName);
85
86          PlayerPrefs.Save();
87
88      }
89
90      //Carga las puntuaciones
91      public void LoadScore()
92      {
93
94          for (int i = 0; i < score0.Length; i++)
95          {
96              score0[i]=PlayerPrefs.GetFloat("Score0" + mapName + i
                    , 0);
97              score1[i] = PlayerPrefs.GetFloat("Score1" + mapName +
                     i, 0);
98              score2[i] = PlayerPrefs.GetFloat("Score2" + mapName +
                     i, 0);
99          }
100
101         CalculateAverages();
102
103     }
```

```
104
105     //Devuelve el mejor tiempo de todas las dificultades
106     public float[] GetBestTime()
107     {
108         float[] times = new float[3];
109         times[0] = score0[0];
110         times[1] = score1[0];
111         times[2]= score2[0];
112
113
114         return times;
115     }
116
117     //Devuelve la media las puntuaciones de todas las
            dificultades
118     public float[] GetAverageTime()
119     {
120         float[] times = new float[3];
121         times[0] = averageScore0;
122         times[1] = averageScore1;
123         times[2] = averageScore2;
124
125
126         return times;
127     }
128
129     //Calcula la media obtenida de todas las dificultades
130     private void CalculateAverages()
131     {
132         //Facil
133         int i=0;
```

```
134
135         averageScore0 = 0;
136         foreach (float num in score0)
137         {
138             if (num != 0)
139             {
140                 i++;
141                 averageScore0 += num;
142             }
143
144         }
145
146         if(i>0)
147             averageScore0 /= i;
148         else
149             averageScore0 = 0;
150
151        //Normal
152        i = 0;
153
154        averageScore1 = 0;
155        foreach (float num in score1)
156        {
157         if (num != 0)
158         {
159             i++;
160             averageScore1 += num;
161         }
162
163        }
164
```

```
165         if(i>0)
166          averageScore1 /= i;
167 
168         else
169          averageScore1 = 0;
170 
171         //Dificil
172          i = 0;
173          averageScore2 = 0;
174          foreach (float num in score2)
175          {
176             if (num != 0)
177              {
178                   i++;
179                   averageScore2 += num;
180              }
181 
182          }
183 
184          if(i>0)
185              averageScore2 /= i;
186          else
187              averageScore2 = 0;
188 
189     }
190 }
```

2.4. MapButton

Este *script* se utiliza representa un botón del selector de mapas. Su función es mostrar información sobre el mapa y permitir acceder a este para comenzar una partida.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using TMPro;
6  using UnityEngine.UI;
7  using System;
8
9  //Botones de la seleccion de mapas
10 public class MapButton : MonoBehaviour
11 {
12     private string mapName; //Nombre del mapa
13     private string sceneName; //Escena donde se encuentra el
       escenario del mapa
14     private float[] bestTime = new float[3]; //Mejor tiempo de
       cada dificultad
15     private float[] averageTime = new float[3]; //Tiempo medio de
        cada dificultad
16     public string MapName
17     {
18         set { mapName = value; }
19     }
20
21     public string SceneName
22     {
```

```
23          set { sceneName = value; }
24      }
25
26      public void SetBestTime(float[] times)
27      {
28          bestTime = times;
29      }
30
31      public void SetAveragteTime(float[] times)
32      {
33          averageTime= times;
34      }
35
36
37      private void Start()
38      {
39          transform.GetChild(0).GetComponentInChildren<TMP_Text>().
                text = mapName;
40
41          var button = GetComponent<Button>();
42          button.onClick.AddListener(OpenMap);
43
44          UpdateDifficulty();
45
46          GameManager.instance.ChangeDifficulty += UpdateDifficulty
                ;
47          GameManager.instance.OnLevelMenu += UpdateDifficulty;
48      }
49
50      //Carga el mapa
51      public void OpenMap()
```

```
52      {
53          //Pantalla de carga
54          FindAnyObjectByType<UIManager>().ActivateLoadingCanvas();
55
56          //Se guarda el ultimo mapa para actualizar las
                puntuaciones
57          PlayerPrefs.SetString("lastMap", mapName);
58          SceneManager.LoadScene(sceneName);
59      }
60
61      //Actualiza la dificultad que se muestra en el boton
62      private void UpdateDifficulty()
63      {
64          int difficulty = PlayerPrefs.GetInt("difficulty",0);
65
66          //Segun la dificultad se lee la mejor puntuacion y la
                media correspondiente
67          switch(difficulty)
68          {
69              case 0:
70                  if (bestTime[0] != 0)
71                  {
72                      transform.GetChild(2).GetComponent<TMP_Text
                            >().text = TimeSpan.FromSeconds(bestTime
                            [0]).ToString(@"mm\:ss");
73                      transform.GetChild(3).GetComponent<TMP_Text
                            >().text = TimeSpan.FromSeconds(averageTime
                            [0]).ToString(@"mm\:ss");
74                  }
75
76                  else
```

```
77                  {
78                      transform.GetChild(2).GetComponent<TMP_Text
                            >().text = "--:--";
79                      transform.GetChild(3).GetComponent<TMP_Text
                            >().text = "--:--";
80                  }
81                  break;
82              case 1:
83                  if (bestTime[1] != 0)
84                  {
85                      transform.GetChild(2).GetComponent<TMP_Text
                            >().text = TimeSpan.FromSeconds(bestTime
                            [1]).ToString(@"mm\:ss");
86                      transform.GetChild(3).GetComponent<TMP_Text
                            >().text = TimeSpan.FromSeconds(averageTime
                            [1]).ToString(@"mm\:ss");
87                  }
88
89                  else
90                  {
91                      transform.GetChild(2).GetComponent<TMP_Text
                            >().text = "--:--";
92                      transform.GetChild(3).GetComponent<TMP_Text
                            >().text = "--:--";
93                  }
94
95                  break;
96              case 2:
97                  if (bestTime[2] != 0)
98                  {
99                      transform.GetChild(2).GetComponent<TMP_Text
```

```
                        >().text = TimeSpan.FromSeconds(bestTime
                        [2]).ToString(@"mm\:ss");
100                 transform.GetChild(3).GetComponent<TMP_Text
                        >().text = TimeSpan.FromSeconds(averageTime
                        [2]).ToString(@"mm\:ss");
101             }
102
103             else
104             {
105                 transform.GetChild(2).GetComponent<TMP_Text
                        >().text = "--:--";
106                 transform.GetChild(3).GetComponent<TMP_Text
                        >().text = "--:--";
107             }
108
109             break;
110         }
111
112     }
113 }
```

2.5. MapManager

Este *script* crea e instancia los *MapButton* a partir de la lista de mapas que tiene el objeto. Esta lista se asigna desde el editor. También se encarga de reajustar las nuevas puntuaciones tras la finalización de una partida.

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.UI;
6  using DG.Tweening;
7  using Unity.VisualScripting;
8  using System.Numerics;
9
10 //Gestiona la informacion de los mapas
11 public class MapManager : MonoBehaviour
12 {
13     [SerializeField] private List<Map> maps = new List<Map>(); //
       Lista con todos los mapas
14     [SerializeField] private GameObject buttonContainer; //
       Contenedor del scrollbar
15     [SerializeField] private MapButton mapButton; //Clase
       MapButton
16     [SerializeField] private GameObject scrollbar; //Scrollbar
17
18     void Start()
19     {
20         GameManager.instance.OnLevelMenu += CreateButton;
21
```

```
22          //Carga las puntuaciones de cada mapa
23          foreach (Map map in maps)
24          {
25              map.LoadScore();
26          }
27      }
28
29      //Crea los botones de la seleccion de mapas
30      private void CreateButton()
31      {
32
33          foreach(Map map in maps)
34          {
35
36              MapButton button;
37              button = Instantiate(mapButton, buttonContainer.
                    transform);
38              button.MapName = map.mapName;
39              button.SceneName = map.sceneName;
40              button.SetBestTime(map.GetBestTime());
41              button.SetAveragteTime(map.GetAverageTime());
42          }
43
44          //Se desuscribre para evitar duplicados
45          GameManager.instance.OnLevelMenu -= CreateButton;
46
47      }
48
49      //Devuelve el numero de mapas
50      public int GetNumberOfMaps()
51      {
```

```
52          return maps.Count;
53      }
54
55      //Recalcula las puntuaciones del ultimo mapa jugado
56      public void RecalculateScores()
57      {
58          //Se recoge la informacion del utlimo mapa guardada
59          string mapName = PlayerPrefs.GetString("lastMap");
60          float newScore = PlayerPrefs.GetFloat("lastScore",-1);
61          int difficulty = PlayerPrefs.GetInt("difficulty");
62
63          //Si la nueva puntuacion no es valida no se actualiza
64          if (newScore == -1)
65              return;
66
67          foreach(Map map in maps)
68          {
69              //Se busca el mapa
70              if (mapName == map.mapName)
71              {
72                  //Se actualiza solo la lista de la dificultad
                        especifica
73                  float[] mapScore = map.GetScoreList(difficulty);
74
75                  //Se inserta de forma ordenada la nueva
                        puntuacion
76                  for (int i=0;i<mapScore.Length;i++)
77                  {
78                      //Se comprueba que la nueva puntuacion es
                            menor o que ya no hay registradas (para
                            evitar actualizar si la nueva puntuacion es
```

```
79                  mayor que las ya registradas)
                    if(newScore < mapScore[i] || mapScore[i]==0)
80                  {
81
82                      if (i>0)
83                      {
84                          //Se compara que sea diferente a la
                               anterior para evitar duplicados
85                          if (newScore != mapScore[i - 1])
86                          {
87
88                              map.SetScoreList(
                                    InsertNewElementInVector(
                                    mapScore, i, newScore),
                                    difficulty);
89                              map.SaveScore();
90
91
92                              return;
93
94                          }
95
96                          //Si es igual no se hace nada
97                          else
98                          {
99
100                             return;
101                         }
102
103
104                     }
```

```
105
106                     //Si es la primera
107                     else
108                     {
109
110                         if (newScore != mapScore[0])
111                         {
112
113
114                             map.SetScoreList(
                                    InsertNewElementInVector(
                                    mapScore, i, newScore),
                                    difficulty);
115                             map.SaveScore();
116
117                             return;
118                         }
119
120                         //Si se obtiene la misma puntuacion
                               no se hace nada
121                         else
122                         {
123
124                             return;
125                         }
126
127                     }
128
129                 }
130             }
131
```

```
132             }
133         }
134     }
135 
136     //Inserta un nuevo elemento en la lista
137     private float[] InsertNewElementInVector(float[] vector, int
        posInsert, float value)
138     {
139         //Deplaza todos los elementos una posicion mas desde el
            final hasta la posicion de insercion
140         for(int i=vector.Length-1;i>posInsert;i--)
141         {
142             vector[i] = vector[i-1];
143         }
144 
145         //Se inserta el nuevo elemento
146         vector[posInsert] = value;
147 
148 
149         return vector;
150     }
151 
152     //Devuelve la lista de mapas
153     public List<Map> GetMaps()
154     {
155         return maps;
156     }
157 }
```

2.6. ScoreButton

Este *script* tiene un propósito similar al *MapButton*. A diferencia de este, su única función es actualizar la tabla de puntuaciones con la del mapa correspondiente cuando se pulsa.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using TMPro;
6  using UnityEngine.UI;
7  using System;
8
9  //Representa los botones del menu de puntuaciones
10 public class ScoreButton : MonoBehaviour
11 {
12     private Map map; //Mapa asociado al boton
13     private GameObject scoresContainer;  //Contenedor del texto
       de puntuaciones
14     public Map Map
15     {
16         set { map = value; }
17     }
18
19     private void Start()
20     {
21         transform.GetChild(0).GetComponentInChildren<TMP_Text>().
           text = map.mapName;
22         scoresContainer = GameObject.FindGameObjectWithTag("
```

```
            Scores");
23
24          var button = GetComponentInChildren<Button>();
25          button.onClick.AddListener(ShowScores);
26      }
27
28      //Muestra todas las puntuaciones registradas en el mapa
29      private void ShowScores()
30      {
31          //Strings para construir el panel de informacion
32          string text0="";
33          string text1="";
34          string text2="";
35
36          //Leemos la lista de puntuaciones
37          float[] score0 = map.GetScoreList(0);
38          float[] score1=map.GetScoreList(1);
39          float[] score2=map.GetScoreList(2);
40
41          //Anadimos iteradamente las puntuaciones
42          for(int i=0;i<score0.Length; i++)
43          {
44              if (score0[i] != 0)
45                  text0 += TimeSpan.FromSeconds(score0[i]).ToString
                        (@"mm\:ss") + "\n";
46
47              else
48                  text0 += "--:--\n";
49
50              if (score1[i]!=0)
51                  text1 += TimeSpan.FromSeconds(score1[i]).ToString
```

```
                (@"mm\:ss")+"\n";
52
53          else
54              text1 += "--:--\n";
55
56          if (score2[i]!=0)
57              text2 += TimeSpan.FromSeconds(score2[i]).ToString
                    (@"mm\:ss")+ "\n";
58          else
59              text2 += "--:--\n";
60      }
61
62      //Si la media no es 0 la anadimos
63      if(map.GetAverageScore(0)!=0)
64          text0 += TimeSpan.FromSeconds(map.GetAverageScore(0))
                .ToString(@"mm\:ss");
65
66      else
67          text0 += "--:--\n";
68
69      if (map.GetAverageScore(1) != 0)
70          text1 += TimeSpan.FromSeconds(map.GetAverageScore(1))
                .ToString(@"mm\:ss");
71
72      else
73          text1 += "--:--\n";
74
75      if (map.GetAverageScore(2) != 0)
76          text2 += TimeSpan.FromSeconds(map.GetAverageScore(2))
                .ToString(@"mm\:ss");
77
```

```
78          else
79              text2 += "--:--\n";
80
81          scoresContainer.transform.GetChild(0).GetComponent<
                TMP_Text>().text=text0;
82          scoresContainer.transform.GetChild(1).GetComponent<
                TMP_Text>().text = text1;
83          scoresContainer.transform.GetChild(2).GetComponent<
                TMP_Text>().text = text2;
84      }
85  }
```

2.7. ScoreManager

Este *script* realiza la misma función de crear e instanciar *ScoreButtons* similar como lo hace la clase *MapManager*.

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.UI;
6  using DG.Tweening;
7  using Unity.VisualScripting;
8  using System.Numerics;
9
10 //Se encarga de crear los botones para el menu de puntuaciones
11 public class ScoreManager : MonoBehaviour
12 {
13     [SerializeField] private GameObject buttonContainer; //
       Contenedor del scrollbar
14     [SerializeField] private ScoreButton scoreButton; //Clase
       ScoreButton
15
16
17     void Start()
18     {
19         //Suscribe la funcion al evento OnScoreMenu
20         GameManager.instance.OnScoreMenu += CreateButton;
21     }
22
23     //Crea los botones del menu de puntuaciones
```

```
24      private void CreateButton()
25      {
26          List<Map> maps = FindAnyObjectByType<MapManager>().
                GetMaps();
27
28          //Por cada mapa instancia un boton
29          foreach(Map map in maps)
30          {
31
32              ScoreButton button;
33              button = Instantiate(scoreButton, buttonContainer.
                    transform);
34              button.Map = map;
35
36          }
37
38          //Es necesario desuscribirlo para no duplicar
39          GameManager.instance.OnScoreMenu -= CreateButton;
40
41      }
42  }
```

2.8. ScrollController

Este *script* se encarga de controlar un *scrollbar* utilizando unos botones de avance y retroceso. En este proyecto se utiliza para desplazar los *MapButtons* y *ScoreButtons*.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  //Controla el funcionamiento del scrollbar
7  public class ScrollController : MonoBehaviour
8  {
9      private int currentStep; //Paso actual
10     private int maxSteps; //Maximo de pasos
11     [SerializeField] RectTransform content; //Contenedor del
           scrollbar
12     Scrollbar scroll;
13     [SerializeField] int offset; //Distancia entre botones
14
15     void Start()
16     {
17         currentStep = 1;
18         maxSteps = FindAnyObjectByType<MapManager>().
               GetNumberOfMaps();
19         scroll = GetComponentInChildren<Scrollbar>();
20
21         //Calcula el paso actual segun la posicion del scroll
22         scroll.onValueChanged.AddListener((value) =>
23         {
```

```
24              float aux = 0;
25              int i = 1;
26
27              while(aux>content.anchoredPosition.x)
28              {
29
30                  aux -= offset/2;
31                  i++;
32              }
33
34              currentStep = Mathf.Clamp(i-2,1,maxSteps);
35          });
36      }
37
38      //Mueve el scroll al siguiente paso, si existe
39      public void Next()
40      {
41          //Centra el scrollbar (para evitar que los botones queden
                desplazados)
42          content.anchoredPosition = (currentStep-1) * new Vector2
               (-offset, 0);
43
44          //Actualiza la posicion del scrollbar
45          if(currentStep < maxSteps)
46          {
47              currentStep++;
48              content.anchoredPosition += new Vector2(-offset, 0);
49          }
50
51      }
52
```

```
53      //Mueve el scroll al anterior paso, si existe
54      public void Previous()
55      {
56          //Centra el scrollbar (para evitar que los botones queden
                desplazados)
57          content.anchoredPosition = (currentStep - 1) * new
               Vector2(-offset, 0);
58
59          //Actualiza la posicion del scrollbar
60          if (currentStep > 1)
61          {
62              currentStep--;
63              content.anchoredPosition += new Vector2(offset, 0);
64
65          }
66      }
67  }
```

2.9. SoundManager

Este *script* se encarga del control y ajuste del volumen de la música y los efectos de sonido de la aplicación. También se encarga de generar algunos sonidos como la música de fondo.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using System.Threading;
4  using Unity.VisualScripting;
5  using UnityEngine;
6  using UnityEngine.UI;
7
8  //Se encarga de controlar el volumen de la musica y los efectos
       de sonido, asi como reproducir la musica y ciertos efectos.
9  public class SoundManager : MonoBehaviour
10 {
11     //Volumen [0-1]
12     private float musicVolume = 1;
13     private float sfxVolume = 1;
14
15     //Botones de silenciar
16     [SerializeField] Image musicButton;
17     [SerializeField] Image sfxButton;
18
19     //Barras de deslizamiento
20     [SerializeField] Slider musicSlider;
21     [SerializeField] Slider sfxSlider;
22
23     //Iconos de silenciar
```

```
24      [SerializeField] Sprite volumeIcon;
25      [SerializeField] Sprite muteIcon;
26
27      private AudioSource audioSource; //Componente encargado del
           sonido
28      [SerializeField] AudioClip alarmClip; //Efecto de alarma
29
30      // Start is called before the first frame update
31      void Start()
32      {
33          //Asignamos los valores predeterminados
34          musicVolume = PlayerPrefs.GetFloat("music", 1);
35          sfxVolume = PlayerPrefs.GetFloat("sfx", 1);
36
37          musicSlider.value= musicVolume;
38          sfxSlider.value= sfxVolume;
39
40          musicSlider.onValueChanged.AddListener((value)=>
               SetMusicVolume(value));
41          sfxSlider.onValueChanged.AddListener((value) =>
               SetSFXVolume(value));
42
43          //En caso de estar en un mapa suscribimos las funciones a
               los eventos
44          if(FindAnyObjectByType<GameplayManager>()!=null)
45          {
46              GameplayManager.instance.OnGameplay += UpdateVolume;
47              GameplayManager.instance.OnGameOver += Alarm;
48          }
49
50          //Activamos la musica
```

```
51          audioSource = GetComponent<AudioSource>();
52          audioSource.loop = true;
53          audioSource.Play();
54
55          UpdateVolume();
56
57
58      }
59
60      //Establece el volumen de la musica
61      public void SetMusicVolume(float value)
62      {
63          if(value<=0)
64          {
65              musicButton.sprite = muteIcon;
66          }
67
68          else
69          {
70              musicButton.sprite = volumeIcon;
71          }
72
73          musicSlider.value = value;
74          musicVolume = value;
75          PlayerPrefs.SetFloat("music", value);
76          UpdateVolume();
77      }
78
79      //Establece el volumen de los efectos de sonido
80      public void SetSFXVolume(float value)
81      {
```

```
82          if (value <= 0)
83          {
84              sfxButton.sprite = muteIcon;
85          }
86
87          else
88          {
89              sfxButton.sprite = volumeIcon;
90          }
91
92          sfxSlider.value = value;
93          sfxVolume= value;
94          PlayerPrefs.SetFloat("sfx", value);
95          UpdateVolume();
96      }
97
98      //Silencia la musica
99      public void MuteMusic()
100     {
101         if(musicVolume>0)
102         {
103             SetMusicVolume(0);
104         }
105
106         else
107         {
108             SetMusicVolume(0.1f);
109         }
110
111         UpdateVolume();
112     }
```

```
113
114     //Silencia los efectos de sonido
115     public void MuteSFX()
116     {
117         if (sfxVolume > 0)
118         {
119             SetSFXVolume(0);
120         }
121
122         else
123         {
124             SetSFXVolume(0.1f);
125         }
126
127         UpdateVolume();
128     }
129
130     //Actualiza todos los componentes que reproducen sonidos
131     private void UpdateVolume()
132     {
133         AudioSource[] audioSources = FindObjectsOfType<
                AudioSource>();
134
135         // Itera a traves de los objetos encontrados
136         foreach (AudioSource audioSource in audioSources)
137         {
138             if(audioSource.gameObject.tag=="music")
139              {
140                  audioSource.volume = musicVolume;
141              }
142
```

```
143             else
144              {
145                  audioSource.volume = sfxVolume;
146              }
147         }
148     }
149
150     //Reproduce una alarma
151     private void Alarm()
152     {
153         audioSource.clip=alarmClip;
154         audioSource.loop = false;
155         audioSource.volume = sfxVolume;
156         audioSource.Play();
157     }
158
159 }
```

2.10. DifficultControl

Este *script* actualiza la dificultad del juego cuando el usuario la selecciona en el menú de selección de mapas.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using static Map;
using TMPro;
using UnityEngine.UI;
using System;

//Controla la dificultad del juego
public class DifficultControl : MonoBehaviour
{
    [SerializeField] private TMP_Text label; //Texto de
       dificultad actual
    private void Start()
    {
        //Lee la ultima dificultad guardada
        int difficulty=PlayerPrefs.GetInt("difficulty",0);

        switch (difficulty)
        {
            case 0: label.SetText("Facil"); break;

            case 1: label.SetText("Normal"); break;

            case 2: label.SetText("Dificil"); break;
```

```
25          }
26
27
28
29      }
30
31      //Cambia la dificultad actual segun la seleccionada en el
            menu de selector de mapas
32      public void ChangeDifficulty()
33      {
34          string difficulty = GetComponentInChildren<TMP_Text>().
                text;
35
36          switch (difficulty)
37          {
38              case "Facil":
39                  PlayerPrefs.SetInt("difficulty", 0);
40                  break;
41              case "Normal":
42                  PlayerPrefs.SetInt("difficulty",1);
43                  break;
44              case "Dificil":
45                  PlayerPrefs.SetInt("difficulty",2);
46                  break;
47          }
48
49          GameManager.instance.Difficulty();
50
51      }
52  }
```

CAPÍTULO 3

PAQUETE GAMEPLAY

Este capitulo expone el codigo fuente de los *scripts* del paquete *Gameplay*. Este paquete recoge los *scripts* que dirigen el comportamiento de una partida.

A continuacion se expondra cada uno:

3.1. GameplayManager

Este *script* sería el equivalente al *GameManager* en una partida. Funciona con eventos también.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using System.Threading;
4  using UnityEngine;
5  using System;
6  using UnityEngine.SceneManagement;
7  using System.Linq;
8
9  //Es la clase principal que dirige la aplicacion durante una
       partida
10 public class GameplayManager : MonoBehaviour
11 {
12     /*
13     * Estos eventos llaman a una serie de funciones suscritas a
           ellos cuando son invocados
14     * permitiendo la comunicacion y el control instantaneo con
           otros scripts
15     */
16     public event Action OnGameplay; //Estado en partida
17     public event Action OnPause; //Estado en pausa
18     public event Action OnFinished; //Estado finalizando partida
19     public event Action OnGameOver; //Estado partida fallida
20     public event Action OnCompleted; //Estado mapa completado
21
22     [SerializeField] string map; //Nombre del mapa
```

```
23      [SerializeField] private int difficulty; //Dificultad actual
24
25      //Tiempo maximo para completar el mapa en cada dificultad
26      [SerializeField] private float timeEasy;
27      [SerializeField] private float timeNormal;
28      [SerializeField] private float timeDifficult;
29
30
31      Timeout timer; //Temporizador
32      private int finishMode; //Modo de finalizacion
33      private string finishModePrefs="finish"; //Clave para leer la
            dificultad
34
35      public static GameplayManager instance; //Instancia del
           script (Patron Singleton)
36      private void Awake()
37      {
38
39          if (instance != null && instance != this)
40          {
41              Destroy(gameObject);
42          }
43
44          else
45          {
46              instance = this;
47          }
48      }
49
50
51      void Start()
```

```
52      {
53          //Se lee la dificultad seleccionada y se inicializa el
                temporizador al tiempo correspondiente
54          difficulty = PlayerPrefs.GetInt("difficulty");
55          timer= FindAnyObjectByType<Timeout>();
56          InitializeObstacles();
57
58          switch(difficulty)
59          {
60              case 0: timer.StartCount(timeEasy); break;
61              case 1: timer.StartCount(timeNormal); break;
62              case 2: timer.StartCount(timeDifficult); break;
63
64          }
65
66          //Se recoge el nombre del mapa
67          map = SceneManager.GetActiveScene().name;
68          OnFinished += SaveData;
69          Gameplay();
70
71      }
72
73      //Activa las funciones necesarias para jugar la partida
74      public void Gameplay()
75      {
76
77          OnGameplay?.Invoke();
78
79      }
80
81      //Activa el menu de pausa
```

```
82      public void Pause()
83      {
84          OnPause?.Invoke();
85
86      }
87
88      //Activa las funciones necesarias para finalizar la partida
89      //Mode = 0 -> Partida completada con exito -- Mode!=0 ->
            Partida fallida
90      public void Finish(int mode)
91      {
92
93          finishMode = mode;
94          OnFinished?.Invoke();
95
96          //Vuelve al menu principal
97          SceneManager.LoadScene("MainScene");
98      }
99
100     //Activa las funciones necesarias cuando una partida se
            completa
101     public void Completed()
102     {
103
104         OnCompleted?.Invoke();
105
106
107     }
108
109     //Activa las funciones necesarias cuando el jugador falla la
            partida
```

```
110     public void GameOver()
111     {
112
113         OnGameOver?.Invoke();
114
115     }
116
117     //Inicializa los obstaculos que varian segun la dificultad
            seleccionada
118     private void InitializeObstacles()
119     {
120         //Si la dificultad==2 por defecto aparecen todos
121         if (difficulty < 2)
122         {
123             GameObject[] obstacles = GameObject.
                    FindGameObjectsWithTag("Obstacles");
124
125             //Para cada obstaculo comprueba si la dificultad
                    actual es igual o superior a su nivel. Si no lo
                    desactiva
126             foreach (GameObject obstacle in obstacles)
127             {
128                 if (obstacle.GetComponent<Obstacle>().Dificulty()
                            > difficulty)
129                 {
130                     obstacle.SetActive(false);
131                 }
132             }
133         }
134
135     }
```

```
136
137     //Reinicia el mapa
138     public void Restart()
139     {
140         SceneManager.LoadScene(map);
141
142     }
143
144     //Guarda como ha finalizado la partida y la puntuacion
        obtenida, si es necesario.
145     private void SaveData()
146     {
147         PlayerPrefs.SetInt(finishModePrefs, finishMode);
148
149         if(finishMode == 0)
150         {
151             PlayerPrefs.SetFloat("lastScore", timer.GetTime());
152         }
153     }
154
155     //Devuelve el modo de finalizacion
156     public int GetFinishMode()
157     {
158         return finishMode;
159     }
160
161     //Devuelve el nombre del mapa
162     public string GetMapName()
163     {
164         return map;
165     }
```

```
166
167     //Devuelve la dificultad actual
168     public int GetDifficulty()
169     {
170         return difficulty;
171     }
172
173 }
```

3.2. Player

Este *script* implementa toda la funcionalidad del personaje jugable durante la partida.

```
1  using DG.Tweening;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Diagnostics;
5  using System.Xml.Serialization;
6  using UnityEngine;
7  using UnityEngine.UI;
8
9  public class Player : MonoBehaviour
10 {
11     [Header("Components")]
12     private Rigidbody2D RB; //Cuerpo del jugador
13
14     [Header("Animator")]
15     private Animator animator; //Controlador de las animaciones
           del jugador
16     private SpriteRenderer SR; //Sprite del jugador
17     private AudioSource audioSource; //Controlador de audio
18     [SerializeField] private AudioClip punchAudio; //Audio
           punetazo
19     [SerializeField] private AudioClip walkAudio; //Audio caminar
20
21
22     [Header("Movement")]
23     [SerializeField] Joystick joystick;
```

```
24      [SerializeField] float moveSpeed; //Velocidad del jugador
25      [SerializeField] float jumpForce; //Fuerza de salto
26      [SerializeField] Transform groundCheckPoint; //Punto de
            deteccion del suelo
27      [SerializeField] LayerMask whatIsGround; //Capa del suelo
28      [SerializeField] LayerMask whatIsTransporter; //Capa del
            suelo
29      private bool isGrounded; //Indica si el jugador esta en el
            suelo
30      private float offset=0; //Offset de velocidad
31      private bool transport = false; //Indica si el personaje esta
             sobre una cinta
32      [SerializeField] Button jumpButton; //Boton de salto
33      private bool isJumping = false; //Indica si el jugador esta
            pulsando el boton de salto
34      [SerializeField] private float jumpTime = 0.2f; //Tiempo de
            espera entre salto
35      private float jumpTimer = 0f; //Contador de tiempo de salto
36
37      [Header ("Incapacite")]
38      [SerializeField] Button incapacitedButton; //Boton de
            incapacitar
39      private BoxCollider2D incapacitedArea; //Rango para
            incapacitar
40      private GameObject enemy; //Enemigo al que incapacita
41
42      [Header ("Others")]
43      public bool visible; //Indica si el jugador es visible para
            los ene migos
44      private bool pause=false;
45      private Vector2 originalVelocity; //Velocidad antes de pausar
```

```
46
47
48      void Start()
49      {
50          //Inicializacion de componentes
51          RB = GetComponent<Rigidbody2D>();
52          animator = GetComponent<Animator>();
53          SR = GetComponent<SpriteRenderer>();
54          audioSource = GetComponent<AudioSource>();
55          incapacitedArea = GetComponentInChildren<BoxCollider2D>()
                ;
56
57          //Se asignan las capas
58          whatIsGround = LayerMask.GetMask("Ground");
59          whatIsTransporter = LayerMask.GetMask("Transport");
60
61          visible = true;
62
63          GameplayManager.instance.OnPause += Pause;
64          GameplayManager.instance.OnGameplay += Play;
65          GameplayManager.instance.OnGameOver += StopSound;
66      }
67
68      void Update()
69      {
70          if(!pause)
71          {
72              //Comprobamos si el jugador esta sobre una cinta de
                    transporte
73              Collider2D collider = Physics2D.OverlapCircle(
                    groundCheckPoint.position, 0.2f, whatIsTransporter)
```

```
            ;
74
75          //Detecta que si
76          if (collider != null)
77          {
78              //Se permite el salto
79              jumpButton.interactable = true;
80              isGrounded = true;
81              transport = true;
82
83              //Se anade un offset a la velocidad dependiendo
                    de hacia donde se mueve la cinta
84              if (collider.transform.localScale.x > 0)
85              {
86                  offset = 0.2f;
87              }
88
89              else
90              {
91                  offset = -0.2f;
92              }
93          }
94
95          else
96          {
97              transport = false;
98
99              //Comprobamos si el jugador esta en el suelo
100             isGrounded = Physics2D.OverlapCircle(
                    groundCheckPoint.position, 0.2f, whatIsGround);
101
```

```
102             //Detecta que si
103             if(isGrounded)
104             {
105                 //Se permite el salto
106                 jumpButton.interactable = true;
107             }
108
109             else
110             {
111                 //No se permite el salto
112                 jumpButton.interactable = false;
113             }
114         }
115
116         //Cuando el personaje aterriza se actualiza el tiempo
                de espera para el siguiente salto
117         if(isGrounded)
118         {
119             UpdateJumpTimer();
120         }
121
122
123         //Obtenemos el input horizontal del controlador
124         float horizontalInput = joystick.Horizontal;
125
126         //Movemos al jugador en el eje X
127         RB.velocity = new Vector2(moveSpeed * (
               horizontalInput + offset), RB.velocity.y);
128
129
130         //Cambiamos el sprite para que concuerde la direccion
```

```
        de movimiento con la direccion del sprite
131     if (RB.velocity.x < 0 && horizontalInput < 0)
132     {
133         SR.flipX = true;
134         incapacitedArea.gameObject.transform.localScale =
                new Vector3(-1, 1, 1);
135     }
136
137     else if (RB.velocity.x > 0)
138     {
139         SR.flipX = false;
140         incapacitedArea.gameObject.transform.localScale =
                new Vector3(1, 1, 1);
141     }
142
143     //Se actualiza el Animator
144     animator.SetFloat("moveSpeed", Mathf.Abs(RB.velocity.
           x));
145     animator.SetBool("isTransporter", transport);
146     animator.SetBool("isGrounded", isGrounded);
147     animator.SetFloat("verticalSpeed", RB.velocity.y);
148
149
150     //Se reinicia el offset para el siguiente frame
151     offset = 0;
152
153     //Se oscurece al personaje si esta oculto
154     if (!visible)
155     {
156         SR.color = new Color(0.77f, 0.77f, 0.77f, 1);
157     }
```

```
158
159             else
160             {
161                 SR.color = new Color(1, 1, 1, 1);
162             }
163
164             //Se comprueba que el boton de salto este pulsado
165             if (isJumping)
166             {
167                 //Se comprueba que paso el tiempo de espera
168                 if(jumpTimer<=0)
169                     Jump();
170             }
171         }
172
173         //Si esta pausado, se mantiene la posicion actual (para
                evitar que caiga por gravedad)
174         else
175         {
176             transform.position = transform.position;
177         }
178
179
180     }
181
182     //Detecta si el jugador esta pulsando el boton de salto
183     public void OnPointerDown()
184     {
185         isJumping = true;
186     }
187
```

```
188     //Detecta si el jugador suelta el boton de salto
189     public void OnPointerUp()
190     {
191         isJumping = false;
192 
193     }
194 
195     //El personaje salta
196     private void Jump()
197     {
198         //Solo puede saltar desde el suelo
199         if(isGrounded)
200         {
201             RB.velocity = new Vector2(RB.velocity.x, jumpForce);
202             jumpTimer = jumpTime;
203         }
204     }
205 
206     //Detecta si un objeto colisiona
207     private void OnTriggerEnter2D(Collider2D collision)
208     {
209         //Si es un obstaculo activa el boton de incapacitar
210         if (collision.CompareTag("Obstacles"))
211         {
212             enemy = collision.gameObject;
213             incapacitedButton.interactable = true;
214         }
215     }
216 
217     //Detecta cuando un objeto sale de la colision
218     private void OnTriggerExit2D(Collider2D collision)
```

```
219     {
220         //Si es un obstaculo desactiva el boton de incapacitar
221         if (collision.CompareTag("Obstacles"))
222         {
223             enemy = null;
224             incapacitedButton.interactable = false;
225         }
226     }
227 
228     //Incapacita a un enemigo cercano
229     public void Incapacitate()
230     {
231 
232         if (enemy != null)
233         {
234             //Animacion de punetazo
235             animator.Play("player_punch", 0);
236 
237             //Sonido de punetazo
238             audioSource.clip = punchAudio;
239             audioSource.pitch = 1f;
240             audioSource.loop = false;
241             audioSource.Play();
242 
243             //Incapacita al enemigo
244             enemy.GetComponent<EnemyController>().Incapacite();
245 
246         }
247 
248     }
249 
```

```
250     //Pausa el personaje
251     private void Pause()
252     {
253         pause = true;
254         animator.speed = 0f; //Pausa la animacion
255         originalVelocity = RB.velocity; //Se almacena la
              velocidad
256         RB.velocity = Vector3.zero;
257         RB.isKinematic = true; //Evita bugs
258     }
259
260     //Reanuda el control del personaje
261     private void Play()
262     {
263         pause = false;
264         RB.isKinematic = false;
265         animator.speed = 1f; //Se restaura la animacion
266         RB.velocity = originalVelocity; //Se restaura la
              velocidad
267     }
268
269     //Para el sonido en ejecucion
270     private void StopSound()
271     {
272         audioSource.Stop();
273     }
274
275     //Devuelve si el personaje esta en el suelo
276     public bool Grounded()
277     {
278         return isGrounded;
```

```
279     }
280
281     //Actualiza el temporizador del tiempo de espera de salto
282     private void UpdateJumpTimer()
283     {
284         jumpTimer -= Time.deltaTime;
285     }
286 }
```

3.3. CameraController

Este *script* controla el comportamiento de la cámara cuando se mueve el personaje. También controla el fondo.

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

//Controlador de la camara
public class CameraController : MonoBehaviour
{
    private Transform target; //Objeto que es seguido por la
       camara
    [SerializeField] float minHeight, maxHeight; //Altura minima
       y maxima
    [SerializeField] float minX, maxX; //Posiciones minima y
       maxima en el eje X
    private Transform farBackground, middleBackground,
       nearBackground; //Fondo
    private Vector2 lastPos; //ultima posicion

    void Start()
    {
        //Se asigna al jugador para que la camara lo siga
        target = GameObject.FindWithTag("Player").GetComponent<
           Transform>();

        lastPos = new Vector2(transform.position.x, transform.
           position.y);
```

```
20
21          farBackground = GameObject.FindGameObjectWithTag("
                FarBackground").GetComponent<Transform>();
22          middleBackground = GameObject.FindGameObjectWithTag("
                MiddleBackground").GetComponent<Transform>();
23          nearBackground = GameObject.FindGameObjectWithTag("
                NearBackground").GetComponent<Transform>();
24      }
25
26      void LateUpdate()
27      {
28
29          //Actualizamos la posicion de la camara a la posicion del
                CameraPoint
30          transform.position = new Vector3(Mathf.Clamp(target.
                position.x, minX, maxX), Mathf.Clamp(target.position.y,
                 minHeight, maxHeight), transform.position.z);
31
32          //Calculamos el desplazamiento del fondo
33          Vector2 amountToMove = new Vector2(transform.position.x -
                lastPos.x, transform.position.y - lastPos.y);
34
35          //Movemos el fondo
36          farBackground.position = farBackground.position + new
                Vector3(amountToMove.x, amountToMove.y, 0f);
37          middleBackground.position += new Vector3(amountToMove.x,
                amountToMove.y, 0f) * 0.5f;
38          nearBackground.position += new Vector3(amountToMove.x,
                amountToMove.y, 0f) * 0.25f;
39
40          //Reiniciamos
```

```
41          lastPos = new Vector2(transform.position.x, transform.
                position.y);
42      }
43  }
```

3.4. Obstacle

Este *script* sirve, en primer lugar, para marcar que objetos del mapa deben activarse según la dificultad. El *GameplayManager* se encarga de esto comprobando que poseen este componente.

Su otra función es activar y deshabilitar algún componente cuando sea necesario, como en el menú pausa.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  /*Controla los obstaculos que deben aparecer segun la dificultad
       */
6  public class Obstacle : MonoBehaviour
7  {
8      [SerializeField] private int dificulty; //Dificultad del mapa
9
10     /*
11      * @brief Funcion que se ejecuta al instanciarse el objeto
12      *
13      */
14     void Start()
15     {
16         GameplayManager.instance.OnGameplay += EnableComponent;
               //Suscribe la funcion EnableComponent al evento
               OnGameplay
17         GameplayManager.instance.OnPause += DisableComponent; //
               Suscribe la funcion DisableComponent al evento OnPause
18         GameplayManager.instance.OnGameOver += DisableComponent;
               //Suscribe la funcion DisableComponent al evento
```

```
        OnGameOver
    }


    public int Dificulty()
    {
        return dificulty;
    }

    private void SetComponentActive(bool enable)
    {
        if (GetComponent<EnemyController>() != null)
        {
            GetComponent<EnemyController>().enabled = enable;
        }

        else if (GetComponentInChildren<SecurityCamera>() != null
            )
        {
            GetComponentInChildren<SecurityCamera>().enabled =
                enable;
        }
    }

    private void EnableComponent()
    {
        SetComponentActive(true);
    }

    private void DisableComponent()
    {
```

```
47          SetComponentActive(false);
48      }
49  }
```

3.5. EnemyController

Este *script* marca el comportamiento de los guardias durante la partida.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using System.Data.SqlTypes;
4  using Unity.VisualScripting;
5  using UnityEngine;
6  using UnityEngine.UIElements;
7
8  //Controlador de los enemigos
9  public class EnemyController : MonoBehaviour
10 {
11     [SerializeField] private float speedMovement; //Velocidad de
          movimiento
12
13     [SerializeField] private Vector2[] pointsMovement; //Vector
          de posiciones de patrulla
14
15     [SerializeField] private float waitTime; //Tiempo de espera
          en cada posicion de patrulla
16
17     [SerializeField] private float startVisionAngle; //angulo
          inicial del area de vision
18
19     private float waitTimer; //Temporizador de espera
20
21     private float incapacitedTimer; //Temporizador de tiempo
          incapacitado
```

```
22
23      private int currentTarget=0; //Posicion objetivo
24
25      private SpriteRenderer sprite;
26
27      private Animator animator;
28
29      private bool incapacitated = false;
30
31      [SerializeField] private float incapacitedTime; //Tiempo que
            permanece incapacitado
32
33      [SerializeField] VisionTriangleController fov; //area de
            vision
34
35      void Start()
36      {
37
38          sprite = GetComponent<SpriteRenderer>();
39          animator=GetComponent<Animator>();
40          Turn();
41
42          fov=GetComponentInChildren<VisionTriangleController>();
43          waitTimer = 0;
44          incapacitedTimer = 0;
45      }
46
47
48      void Update()
49      {
50          if (!incapacitated)
```

```
51          {
52
53
54                  if (MoveToTarget())
55                  {
56                      //Se desplaza hacia la posicion
                           correspondiente
57                      transform.position = Vector2.MoveTowards(
                           transform.position, pointsMovement[
                           currentTarget], speedMovement * Time.
                           deltaTime);
58                      animator.SetBool("isWalking", true);
59                  }
60
61                  else
62                  {
63                      //Espera un tiempo en la posicion actual
64                      UpdateWaitTimer();
65                      animator.SetBool("isWalking", false);
66
67                      //Cuando finaliza el tiempo de espera se
                           avanza a la siguiente posicion
68                      if (waitTimer >= waitTime)
69                      {
70                          currentTarget = (currentTarget + 1) %
                               pointsMovement.Length;
71                          Turn();
72                          waitTimer = 0;
73                      }
74                  }
75
```

```
76
77
78              //Dibuja el area de vision
79              fov.DrawMesh();
80          }
81
82          else
83          {
84              //Actualiza el temporizador
85              UpdateIncapacitedTimer();
86
87              //Se despierta al alcanzar el tiempo maximo
                    incapacitado
88              if(incapacitedTimer >= incapacitedTime)
89              {
90                  WakeUp();
91              }
92          }
93
94      }
95
96      //Comprueba si tiene que desplazarse
97      private bool MoveToTarget()
98      {
99          //Si no ha alcanzado la posicion objetivo y no esta
                incapacitado, tiene que continuar moviendose
100         if(Vector2.Distance(pointsMovement[currentTarget],
                transform.position)>0.1 && !incapacitated)
101         {
102             return true;
103         }
```

```
104
105         return false;
106     }
107
108     //Ajusta el sentido del sprite y el area de vision
109     private void Turn()
110     {
111         //El personaje se mueve hacia la derecha
112         if (transform.position.x < pointsMovement[currentTarget].
                x)
113         {
114             //Se actualiza el angulo inicial
115             fov.setInitialAngle(startVisionAngle);
116
117             //Se ajusta el punto de origen del fov
118             transform.GetChild(0).GetComponent<Transform>().
                    position = transform.position + new Vector3(0.3f,
                    -0.28f,0);
119
120             //Se ajusta el sprite
121             sprite.flipX = false;
122         }
123
124         //El personaje se mueve hacia la izquierda
125         else
126         {
127             fov.setInitialAngle(startVisionAngle+180);
128             transform.GetChild(0).GetComponent<Transform>().
                    position = transform.position + new Vector3(-0.3f,
                    -0.28f, 0);
129             sprite.flipX = true;
```

```
130         }
131     }
132     //Actualiza el temporizador del tiempo de espera
133     private void UpdateWaitTimer()
134     {
135         waitTimer += Time.deltaTime;
136     }
137 
138     //Actualiza el temporizador del tiempo incapacitado
139     private void UpdateIncapacitedTimer()
140     {
141         incapacitedTimer += Time.deltaTime;
142     }
143 
144     //Incapacita al personaje
145     public void Incapacite()
146     {
147 
148         //Evita que el jugador pueda incapacitar infinitamente
149         if (!incapacitated)
150         {
151             animator.SetBool("Incapacited", true);
152             incapacitated = true;
153             incapacitedTimer = 0;
154             fov.gameObject.SetActive(false);
155 
156         }
157     }
158 
159     //El personaje despierta despues de haber sido incapacitado
160     private void WakeUp()
```

```
161     {
162         animator.SetBool("Incapacited", false);
163 
164         //Espera a que finalice la animacion de levantarse
165         if(animator.GetCurrentAnimatorStateInfo(0).IsName("
                enemy_idle"))
166         {
167             incapacitated = false;
168             fov.gameObject.SetActive(true);
169         }
170 
171     }
172 }
```

3.6. SecurityCamera

Este *script* marca el funcionamiento de las cámaras de seguridad durante la partida.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using static UnityEngine.GraphicsBuffer;
5
6  //Controlador de las camaras de seguridad
7  public class SecurityCamera : MonoBehaviour
8  {
9      [SerializeField] Quaternion finalRotation; //Rotacion final
           de la cabeza
10     [SerializeField] float rotationSpeed; //Velocidad de rotacion
11     [SerializeField] VisionTriangleController fov; //area de
           vision
12
13     private Quaternion startRotation; //Rotacion inicial (por
           defecto la que tiene en la escena)
14     private Quaternion targetRotation; //Rotacion objetivo (por
           defecto finalRotation)
15     private float progress = 0f; //Indica el progreso de rotacion
16
17     [SerializeField] private float waitTime; //Tiempo de espera
18
19     private float timer; //Contador para el tiempo de espera
20
21
22     void Start()
```

```
23      {
24          startRotation = transform.rotation; // Rotacion inicial
                del objeto
25          targetRotation = finalRotation; // Rotacion objetivo del
                objeto
26          timer = 0;
27
28      }
29
30      private void Update()
31      {
32          // Incrementar el progreso de la rotacion
33          progress += rotationSpeed * Time.deltaTime;
34
35          // Interpolar la rotacion actual entre la rotacion
                inicial y la rotacion objetivo
36          transform.rotation = Quaternion.Slerp(startRotation,
                targetRotation, progress);
37
38          // Si se ha alcanzado la rotacion objetivo, reiniciar la
                rotacion
39          if (progress >= 1f)
40          {
41              UpdateTimer();
42
43              if (timer>waitTime)
44              {
45                  progress = 0f;
46
47                  //Ahora la rotacion inicial y objetivo se
                        intercambian para revertir el movimiento
```

```
48                  Quaternion tempRotation = startRotation;
49                  startRotation = targetRotation;
50                  targetRotation = tempRotation;
51                  timer = 0;
52              }
53
54          }
55
56          //Se redibuja el mesh con la rotacion actual
57          else
58             fov.DrawMesh(transform.rotation.z*100);
59
60
61      }
62
63      //Actualiza el contador
64      private void UpdateTimer()
65      {
66          timer += Time.deltaTime;
67      }
68
69  }
```

3.7. VisionTriangleController

Este *script* se encarga de dibujar y renderizar el área de visión de los enemigos, así como de detectar al jugador

```
1  using System.Collections.Generic;
2  using UnityEngine;
3  using Unity.VisualScripting;
4  using System.Net.NetworkInformation;
5  using System;
6  using UnityEngine.UIElements;
7
8  [RequireComponent(typeof(PolygonCollider2D))]
9
10 //Dibuja y controla un area de vision en forma de triangulo
11 public class VisionTriangleController : MonoBehaviour
12 {
13     [SerializeField] private float viewDistance = 5f; //Distancia
           que alcanza de vision
14     [SerializeField] private float viewAngle = 90f; //angulo que
          puede detectar
15     [SerializeField] private float initialAngle = 90f; //angulo
          de posicion inicial
16     [SerializeField] private LayerMask obstacleLayerMask; //Capa
          que detecta el suelo
17     [SerializeField] private int segments = 30; // Numero de
          segmentos para el mesh
18     [SerializeField] private Transform originPoint; //Punto de
          origen para dibujar
19     [SerializeField] private Material fovMaterial; //Material
```

```
20      [SerializeField] private Material fovMaterialDetection; //
            Material al detectar
21      private float  timeDetection=0; //Tiempo transcurrido
            detectando al jugador
22      public float startAngle; //angulo inicial
23      private Mesh visionMesh; //Mesh que representa el area
24      private MeshFilter visionMeshFilter;
25      private MeshRenderer visionMeshRenderer;
26      private PolygonCollider2D polygonCollider; //Collider del
            Mesh
27      public bool isPlayerDetected; //Informa si el jugador ha sido
             detectado
28      public bool isFinish; //Informa si la partida finaliza
29      void Start()
30      {
31          visionMesh = new Mesh();
32          visionMeshFilter = GetComponent<MeshFilter>();
33          visionMeshRenderer = GetComponent<MeshRenderer>();
34          polygonCollider = GetComponent<PolygonCollider2D>();
35          isPlayerDetected = false;
36          isFinish = false;
37      }
38
39      //Se activa cuando un objeto entra al collider
40      private void OnTriggerEnter2D(Collider2D collider)
41      {
42          //Si el collider detectado es el jugador
43          if (collider.CompareTag("Player"))
44          {
45              //Si el jugador es visible (no esta escondido)
46              if (collider.gameObject.GetComponent<Player>().
```

```
            visible)
47          {
48
49              isPlayerDetected = true;
50
51              visionMeshRenderer.material =
                    fovMaterialDetection;
52
53              timeDetection = 0;
54          }
55
56      }
57  }
58
59  //Se activa cuando un objeto se mantiene dentro del collider
60  private void  OnTriggerStay2D(Collider2D collider)
61  {
62      //Si el collider detectado es el jugador
63      if (collider.CompareTag("Player"))
64      {
65          //Si el jugador es visible (no esta escondido)
66          if (collider.gameObject.GetComponent<Player>().
                visible)
67          {
68              //Actualizamos el contador
69              timeDetection += Time.deltaTime;
70
71              //Si ha pasado suficiente tiempo detectado o el
                    jugador esta muy cerca del origen del area se
                    considera que ha sido detectado y pierde la
                    partida
```

```
72                  if (timeDetection > 0.1f || Vector2.Distance(
                        collider.transform.position, originPoint.
                        transform.position)<0.1f)
73                  {
74                      //Se congela al enemigo
75                      collider.GetComponent<Rigidbody2D>().
                            constraints = RigidbodyConstraints2D.
                            FreezeAll;
76                      //Se desactiva al jugador
77                      collider.GetComponent<Player>().enabled =
                            false;
78                      collider.GetComponent<Animator>().SetBool("
                            finish", true);
79
80                      //Se llama al evento GameOver
81                      GameplayManager.instance.GameOver();
82
83                  }
84              }
85
86
87          }
88
89      }
90
91      //Se activa cuando un objeto sale del collider
92      private void OnTriggerExit2D(Collider2D collider)
93      {
94          //Si es el jugador
95          if (collider.CompareTag("Player"))
96          {
```

```
97              //Se regresa al estado no detectado
98              isPlayerDetected = false;
99
100             visionMeshRenderer.material = fovMaterial;
101
102             timeDetection = 0;
103         }
104     }
105
106     //Asigna un angulo inicial
107     public void setInitialAngle(float angle)
108     {
109         initialAngle = angle;
110     }
111
112     //Dibuja el mesh con una cierta rotacion
113     public void DrawMesh(float rotation=0)
114     {
115         transform.position = originPoint.position;
116
117         startAngle = initialAngle - rotation;
118
119         // Crear un nuevo mesh para el area de vision
120         visionMesh.Clear();
121         Vector3[] vertices = new Vector3[segments + 1 + 1]; //
               Vertices del mesh
122         Vector2[] points = new Vector2[vertices.Length]; //Puntos
               del collider
123         int[] triangles = new int[segments * 3]; //Triangulos del
               mesh
124
```

```
125         // Calcular los vertices del mesh y detectar objetos
126         Vector3 origin = transform.position;
127         vertices[0] = Vector3.zero;
128         points[0] = Vector2.zero;
129         float angleIncrement = viewAngle / segments; //angulo de
               diferencia entre cada segmento
130         float currentAngle = -viewAngle / 2f - startAngle; //
               angulo inicial
131
132
133         int vertexIndex = 1;
134         int triangleIndex = 0;
135
136         for (int i = 0; i <= segments; i++)
137         {
138             Vector3 vertex;
139             Vector3 direction = Quaternion.Euler(0, 0,
                   currentAngle) * transform.up; //Calcula la
                   direccion segun el angulo
140
141             RaycastHit2D hit = Physics2D.Raycast(origin,
                   direction, viewDistance, obstacleLayerMask); //
                   Detecta si hay colision
142             if (hit.collider != null)
143             {
144                 //Si se detecta colision el vertice se dibuja en
                       la colision
145                 vertex = transform.InverseTransformPoint(hit.
                       point);
146             }
147             else
```

```
148             {
149                 //Si no se detecta colision el vertice se dibuja
                        en la distancia maxima
150                 vertex = transform.InverseTransformPoint(origin +
                        direction * viewDistance);
151             }
152
153
154             //Se anade el nuevo vertice
155             vertices[vertexIndex] = vertex;
156             points[vertexIndex] = vertex;
157
158             // Se asignan los vertices a cada triangulo
159             if (i > 0)
160             {
161                 triangles[triangleIndex] = 0;
162                 triangles[triangleIndex + 1] = vertexIndex - 1;
163                 triangles[triangleIndex + 2] = vertexIndex;
164
165                 triangleIndex += 3;
166             }
167
168             vertexIndex++;
169             currentAngle -= angleIncrement;
170
171         }
172
173         // Asignar los vertices y triangulos al mesh
174         visionMesh.vertices = vertices;
175         visionMesh.triangles = triangles;
176
```

```
177         //Asignar los puntos al collider
178         polygonCollider.points = points;
179 
180         // Recalcular las normales y limites del mesh
181         visionMesh.RecalculateNormals();
182         visionMesh.RecalculateBounds();
183 
184         // Asignar el mesh actualizado al MeshFilter
185         visionMeshFilter.mesh = visionMesh;
186     }
187 }
```

3.8. SafePoint

Este implementa la funcionalidad de esconderse detrás de un objeto para evitar ser detectado.

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

//Representa los objetos que sirven al jugador para esconderse
public class SafePoint : MonoBehaviour
{
    private UIGameplayManager UIgameManager; //Controlador de la
       interfaz

    void Start()
    {
        UIgameManager=FindAnyObjectByType<UIGameplayManager>();
    }

    // Evento que se activa al detectar una colision
    private void OnTriggerEnter2D(Collider2D other)
    {
        //Si la colision es el jugador, se da el efecto de estar
           oculto y el personaje no es visible a los enemigos
        if(other.CompareTag("Player"))
        {
            other.gameObject.GetComponent<Player>().visible =
               false;
            UIgameManager.ActivatePanelSafePoint(true);
```

```
23          }
24      }
25
26      // Evento que se activa al finalizar una colision
27      private void OnTriggerExit2D(Collider2D other)
28      {
29          //Si la colision es el jugador, se elimina el efecto de
                estar oculto y el personaje es visible a los enemigos
30          if (other.CompareTag("Player"))
31          {
32              other.gameObject.GetComponent<Player>().visible =
                    true;
33              UIgameManager.ActivatePanelSafePoint(false);
34          }
35      }
36  }
```

3.9. FinishPoint

Este *script* representa el final del mapa y se encarga de ejecutar la animación final de la partida.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  //Representa el final del mapa y controla todo lo necesario para
       finalizar una partida completada
6  public class FinishPoint : MonoBehaviour
7  {
8      //Animadores
9      [SerializeField] Animator screen;
10     [SerializeField] Animator door;
11
12     //Sonidos
13     [SerializeField] AudioClip screenClip;
14     [SerializeField] AudioClip doorClip;
15
16     AudioSource audioSource;
17     private int phase = 0; //Variable que controla el orden de
           accion
18
19
20     private void Start()
21     {
22         audioSource = GetComponent<AudioSource>();
23     }
```

```
24      private void OnTriggerStay2D(Collider2D collision)
25      {
26          //Si el jugador es detectado
27          if(collision.CompareTag("Player"))
28          {
29              //Se congela al jugador
30              GameObject player = FindFirstObjectByType<Player>().
                    gameObject;
31              player.GetComponent<Rigidbody2D>().constraints =
                    RigidbodyConstraints2D.FreezePositionX; //Solo se
                    congela el eje X para que el personaje caiga al
                    suelo
32              player.GetComponent<Rigidbody2D>().constraints =
                    RigidbodyConstraints2D.FreezeRotation;
33
34              //Si el jugador no esta saltando
35              if (phase==0 && player.GetComponent<Player>().
                    Grounded())
36              {
37                  //Se desactiva el control del jugador
38                  player.GetComponent<Rigidbody2D>().constraints =
                        RigidbodyConstraints2D.FreezeAll; // Se congela
                         ahora todos los ejes
39                  player.GetComponent<Player>().enabled = false;
40                  player.GetComponent<Animator>().SetBool("finish",
                         true);
41
42                  //Animacion de la pantalla
43                  screen.SetBool("Finish", true);
44                  audioSource.clip = screenClip;
45                  audioSource.Play();
```

```
46                  phase = 1;
47              }
48
49
50          }
51      }
52
53
54      private void Update()
55      {
56          //Cuando se ha realizado la animacion de la pantalla
57          if(phase==1 && screen.GetCurrentAnimatorStateInfo(0).
                IsName("screen_finish"))
58          {
59              //Animacion de la puerta
60              audioSource.clip = doorClip;
61              audioSource.Play();
62              door.SetBool("Finish", true);
63              phase = 2;
64
65          }
66
67          //Cuando se ha realizado la animacion de la puerta
68          else if(phase==2 && door.GetCurrentAnimatorStateInfo(0).
                IsName("door_finish"))
69          {
70              //Se finaliza la partida
71              GameplayManager.instance.Completed();
72          }
73
74      }
```

```
75 }
```

3.10. Timeout

Este *script* es el temporizador de la partida. Su única función es marcar el tiempo restante.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using TMPro;
5  using System;
6
7  //Temporizador del mapa
8  public class Timeout:MonoBehaviour
9  {
10     [SerializeField] private float time=0;
11     private bool enable = false;
12     [SerializeField] private TMP_Text text; //Texto donde se
          imprime el temporizador
13     private string timeText; //Temporizador en formato de texto
14     private float maxTime; //Tiempo maximo para finalizar el mapa
15     private void Start()
16     {
17         GameplayManager.instance.OnGameplay += ReanudeCount;
18         GameplayManager.instance.OnPause += PauseCount;
19         GameplayManager.instance.OnFinished += PauseCount;
20         GameplayManager.instance.OnCompleted += PauseCount;
21     }
22
23     //Inicializa y activa el temporizador
24     public void StartCount(float MaxTime)
25     {
```

```
26          time = MaxTime;
27          maxTime = MaxTime;
28          enable = true;
29      }
30
31
32      private void Update()
33      {
34          //Actualiza el tiempo si esta activo
35          if(enable)
36          {
37              time -= Time.deltaTime;
38
39              //Si el tiempo llega a cero se da por finalizada la
                    partida
40              if(time <= 0 )
41              {
42                  enable = false;
43                  GameplayManager.instance.GameOver();
44              }
45          }
46
47          //Imprime el temporizador en la pantalla
48
49          timeText = TimeSpan.FromSeconds(time).ToString(@"mm\:ss")
                ;
50
51          text.text = timeText;
52
53
54      }
```

```
55
56      //Pausa el temporizador
57      public void PauseCount()
58      {
59          enable = false;
60      }
61
62      //Reanuda el temporizador
63      public void ReanudeCount()
64      {
65          enable = true;
66      }
67
68      //Devuelve el tiempo empleado
69      public float GetTime()
70      {
71          return maxTime-time;
72      }
73
74      //Comprueba si el tiempo ha expirado
75      public bool TimeHasExpired()
76      {
77          if (time <= 0)
78              return true;
79
80          return false;
81      }
82
83  }
```

3.11. UIGameplayManager

Este *script* se encarga del control de la interfaz como la clase *UIManager*.

```
using DG.Tweening;
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using static Unity.Burst.Intrinsics.X86.Sse4_2;

//Controla la interfaz dentro de una partida
public class UIGameplayManager : MonoBehaviour
{
    //Canvas de cada menu
    [SerializeField] GameObject gameplayCanvas;
    [SerializeField] GameObject pauseCanvas;
    [SerializeField] GameObject gameOverCanvas; //Se activa
       cuando el usuario pierde la partida
    [SerializeField] GameObject finishCanvas; //Se activa cuando
       el usuario gana la partida

    private void Start()
    {
        GameplayManager.instance.OnGameplay +=
           ActivateGameplayCanvas;
        GameplayManager.instance.OnPause += ActivatePauseCanvas;
        GameplayManager.instance.OnGameOver +=
           ActivateGameOverCanvas;
```

```
23          GameplayManager.instance.OnCompleted +=
                ActivateFinishCanvas;
24      }
25
26      //Activa la interfaz de juego
27      private void ActivateGameplayCanvas()
28      {
29          gameplayCanvas.SetActive(true);
30          pauseCanvas.SetActive(false);
31      }
32
33      //Activa el menu de pausa
34      private void ActivatePauseCanvas()
35      {
36          gameplayCanvas.SetActive(false);
37          pauseCanvas.SetActive(true);
38
39          //Se lee el mejor tiempo registrado del mapa en la
                dificultad actual
40          float bestTime = PlayerPrefs.GetFloat("Score"+
                GameplayManager.instance.GetDifficulty() + PlayerPrefs.
                GetString("lastMap")+ "0");
41
42          //Si es valido
43          if(bestTime > 0)
44          {
45              //Se muestra el texto
46              pauseCanvas.transform.GetChild(5).transform.GetChild
                    (0).GetComponentInChildren<TMP_Text>().text =
                    TimeSpan.FromSeconds(bestTime).ToString(@"mm\:ss")
                    + "\n";
```

```
47          }
48
49          //Se muestra la puntuacion actual
50          pauseCanvas.transform.GetChild(4).transform.GetChild(0).
                GetComponent<TMP_Text>().text = TimeSpan.FromSeconds(
                FindAnyObjectByType<Timeout>().GetTime()).ToString(@"mm
                \:ss") + "\n";
51      }
52
53      //Activa el menu de partida perdida
54      private void ActivateGameOverCanvas()
55      {
56          gameplayCanvas.SetActive(false);
57          gameOverCanvas.SetActive(true);
58
59          //Comprueba por que ha finalizado la partida y se imprime
                el mensaje correspondiente
60          if (FindAnyObjectByType<Timeout>().TimeHasExpired())
61          {
62              gameOverCanvas.transform.GetChild(1).GetComponent<
                    TMP_Text>().text = "SE HA AGOTADO EL TIEMPO";
63          }
64
65          else
66          {
67              gameOverCanvas.transform.GetChild(1).GetComponent<
                    TMP_Text>().text = "HAS SIDO DETECTADO";
68          }
69
70
71          gameOverCanvas.transform.GetChild(1).transform.DOScale(
```

```
            new Vector3(1, 1, 1), 1f);
72
73          //Se da algo de delay para que aparezcan los botones
74          StartCoroutine(Pausar());
75          gameOverCanvas.transform.GetChild(2).transform.DOScale(
               new Vector3(1.5f, 1.5f, 1), 2f);
76          gameOverCanvas.transform.GetChild(3).transform.DOScale(
               new Vector3(1.5f, 1.5f, 1), 2f);
77      }
78
79      //Activa el menu de partida completada
80      private void ActivateFinishCanvas()
81      {
82
83          gameplayCanvas.SetActive(false);
84          finishCanvas.SetActive(true);
85
86          //Se lee el mejor tiempo registrado del mapa en la
               dificultad actual
87          float bestTime = PlayerPrefs.GetFloat("Score0" +
               PlayerPrefs.GetString("lastMap") + "0");
88
89          //Si es valido se imprime
90          if (bestTime > 0)
91          {
92              finishCanvas.transform.GetChild(4).transform.GetChild
                   (0).GetComponentInChildren<TMP_Text>().text =
                   TimeSpan.FromSeconds(bestTime).ToString(@"mm\:ss")
                   + "\n";
93          }
94
```

```
95          //Se lee la puntuacion de la partida y se imprime
96          float newTime = FindAnyObjectByType<Timeout>().GetTime();
97
98          finishCanvas.transform.GetChild(3).transform.GetChild(0).
               GetComponent<TMP_Text>().text = TimeSpan.FromSeconds(
               newTime).ToString(@"mm\:ss") + "\n";
99
100         //Si la mejor puntuacion no es valida o es peor que la
               nueva, la nueva puntuacion sustituye a la antigua y se
               resalta en amarillo
101         if(bestTime == 0 || bestTime>newTime)
102         {
103             finishCanvas.transform.GetChild(4).transform.GetChild
                   (0).GetComponentInChildren<TMP_Text>().text =
                   TimeSpan.FromSeconds(newTime).ToString(@"mm\:ss") +
                    "\n";
104             finishCanvas.transform.GetChild(4).transform.GetChild
                   (0).GetComponentInChildren<TMP_Text>().color =
                   Color.yellow;
105
106         }
107     }
108
109     //Funcion que espera 3 segundos
110     IEnumerator Pausar()
111     {
112         yield return new WaitForSecondsRealtime(3f);
113     }
114
115     //Activa el panel que oscurece la pantalla para dar el efecto
           de estar escondido
```

```
116     public void ActivatePanelSafePoint(bool enable)
117     {
118         gameplayCanvas.transform.GetChild(0).gameObject.SetActive
                (enable);
119     }
120 }
```