

Desarrollo de un juego sobre gestión de proyectos software

TRABAJO DE FIN DE GRADO

AUTOR: MARIO DÍAZ SANTOS

DIRECTOR: CARLOS BLANCO BUENO

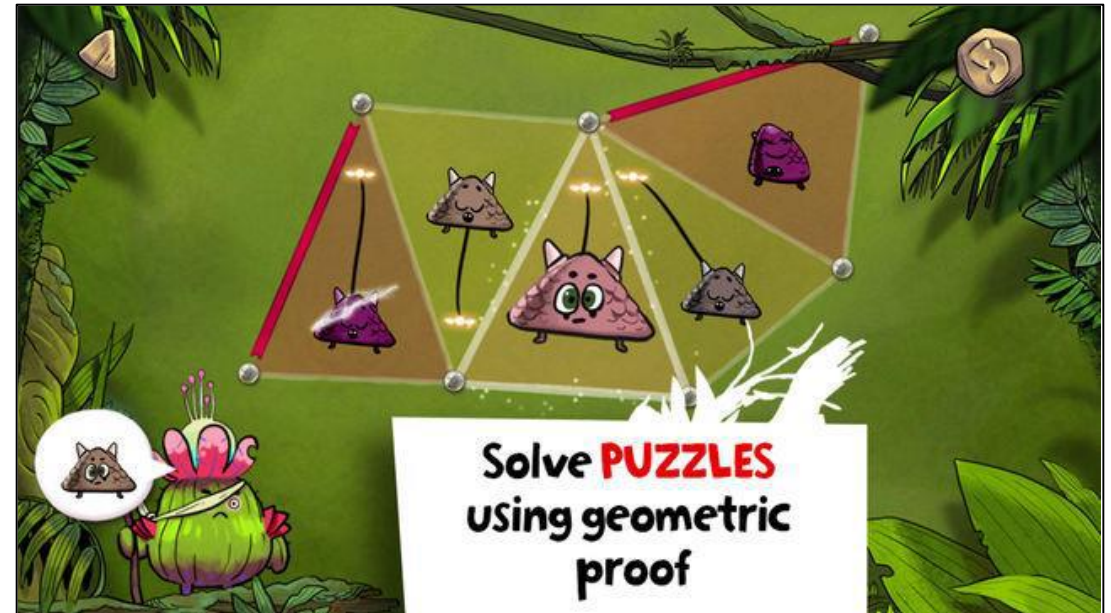


ÍNDICE

- **Introducción**
- Herramientas y metodología
- Requisitos
- Diseño e implementación
- Pruebas
- Demo
- Conclusiones

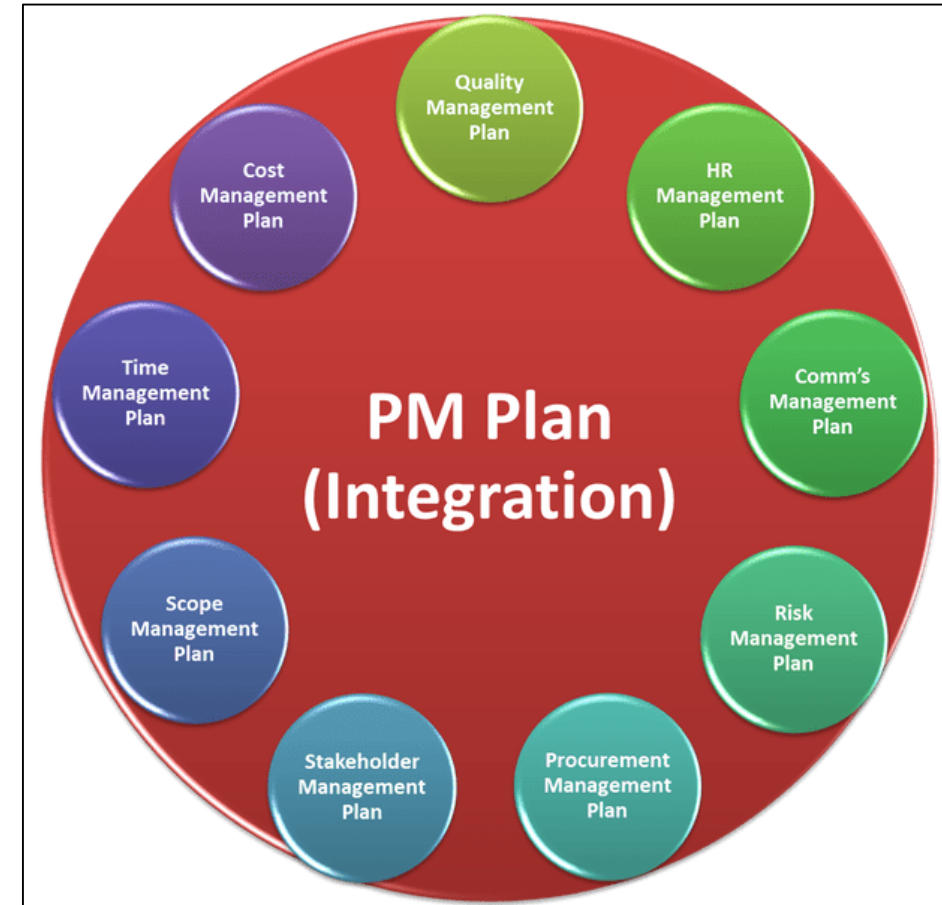
1. INTRODUCCIÓN

- Contexto: *Serious Games*
 - ¿Qué son?
 - Elementos que garantizan el éxito:
 - Historia.
 - Gamificación.
 - Feedback inmediato.
 - Simulación.
 - Aprendizaje como objetivo.
 - *Dragon Box Elements*



1. INTRODUCCIÓN

- Contexto: Gestión de proyectos
 - Planificar y controlar los aspectos relevantes durante el desarrollo de un proyecto.
 - PMBOK
 - Elementos gestionados:
 - Costes.
 - Tiempo.
 - Alcance.
 - Riesgos.
 - ...



1. INTRODUCCIÓN

- Contexto: *Deliver*



Risk
C. Wangenheim CreativeCommons©2010

An investor became seriously interested in your project and started to provide additional capital.

Take \$ 1.000 from the bank and advance the sum of the productivity factors of your team.

Human Resource

Bill



Productivity 3
Weekly salary \$ 1.000

C. Wangenheim CreativeCommons©2010

1. INTRODUCCIÓN

- Motivación

- Creación de un juego serio para gestión de proyectos.
- Utilización de nuevas tecnologías:
 - Un motor de desarrollo de videojuegos.
 - Programas de desarrollo gráfico.
- Realización de un producto software completo siguiendo una metodología clara.

1. INTRODUCCIÓN

- Objetivo

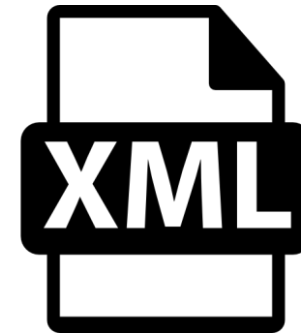
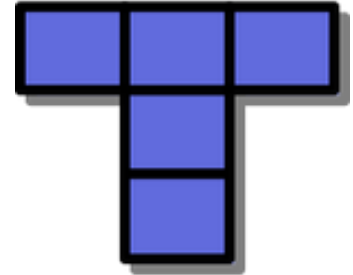
- La versión física del juego tiene algunos inconvenientes:
 - Problemas logísticos.
 - Rígido uso de los materiales y necesidad de tenerlos accesibles (tablero, fichas, dado...).
- La implementación de una versión digital soluciona los inconvenientes.
- Además, permite la introducción de mejoras como:
 - Distintas configuraciones para las reglas del juego.
 - Variabilidad en los recursos (tablero, fichas de recursos y fichas de empleados).
 - Multidispositivo.
 - Modo multijugador.

ÍNDICE

- Introducción
- **Herramientas y metodología**
- Requisitos
- Diseño e implementación
- Pruebas
- Demo
- Conclusiones

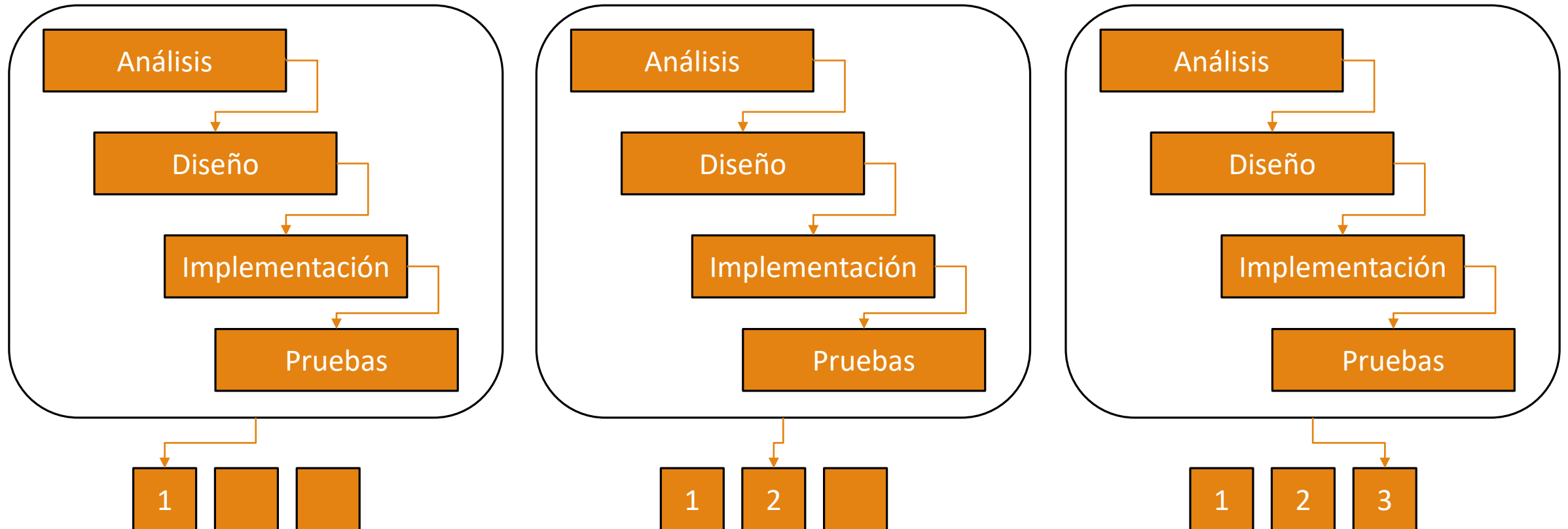
2. HERRAMIENTAS Y METODOLOGÍA

- Software y tecnologías utilizados



2. HERRAMIENTAS Y METODOLOGÍA

- Metodología iterativa incremental



2. HERRAMIENTAS Y METODOLOGÍA

- Plan de proyecto
 - 3 etapas:
 - Formación.
 - Desarrollo.
 - Integración.
 - Duración total: 130 días.

ETAPAS	FECHA DE INICIO	FECHA DE FIN
ETAPA DE FORMACIÓN	08/10/2018	21/10/2018
ITERACIÓN 1	22/10/2018	04/11/2018
ITERACIÓN 2	05/11/2018	11/11/2018
ITERACIÓN 3	12/11/2018	18/11/2018
ITERACIÓN 4	19/11/2018	25/11/2018
ITERACIÓN 5	26/11/2018	02/12/2018
ITERACIÓN 6	03/12/2018	16/12/2018
ITERACIÓN 7	17/12/2018	30/12/2018
ITERACIÓN 8	31/12/2018	06/01/2019
ITERACIÓN 9	07/01/2019	20/01/2019
ITERACIÓN 10	21/01/2019	27/01/2019
ETAPA DE INTEGRACIÓN	28/01/2019	15/02/2019

ÍNDICE

- Introducción
- Herramientas y metodología
- **Requisitos**
- Diseño e implementación
- Pruebas
- Demo
- Conclusiones

3. REQUISITOS

● Requisitos funcionales

○ Iteración 1

- RF06, RF08, RF14.

○ Iteración 2

- RF03, RF14.

○ Iteración 3

- RF03, RF16.

○ Iteración 4

- RF03, RF06, RF08, RF10, RF12, RF14.

○ Iteración 5

- RF09, RF10, RF15, RF17, RF19.

#	DESCRIPCIÓN
RF01	El juego deberá ser jugado en modo multijugador.
RF02	El juego permitirá configurar parámetros iniciales como: el número de empleados iniciales o el tablero de entre los disponibles.
RF03	El juego se desarrollará por turnos.
RF04	El juego automatizará el cálculo de valores tales como el SPI, CPI o el coste total de una fase a partir de unos parámetros introducidos por el jugador.
RF05	El juego continuará pese a que un jugador abandone la partida.
RF06	El juego contará con un dado como método para que el jugador avance las casillas correspondientes.
RF07	El juego finalizará cuando un jugador gane, todos se queden sin dinero menos uno o todos, menos uno, abandonen la partida.
RF08	El tablero tendrá una casilla de salida y una de llegada o meta.
RF09	Al finalizar la partida, el sistema mostrará un sumario con la posición de cada jugador en la partida.
RF10	Cuando un jugador llegue a un hito, el sistema mostrará el reporte de actuación de la fase pasada, la ventana de asignación de empleados y el plan de proyecto para la siguiente fase.
RF11	El jugador deberá introducir, en el reporte de actuación de la fase, el valor ganado y el sistema comprobará que el valor es correcto. Después, el sistema mostrará los valores del CPI y el SPI.
RF12	El jugador, utilizando la ventana de asignación de empleados, deberá seleccionar los empleados que desea tener disponibles para la siguiente fase.
RF13	El jugador deberá introducir una nueva planificación para la siguiente fase al inicio y al llegar a un hito.
RF14	El sistema permitirá tener acceso en todo momento a la información relevante del proyecto como el plan de proyecto, el reporte de actuación, los empleados o el dinero disponible.
RF15	Si se lanza el dado y sale un número entre 1 y 4 el sistema avanzará la ficha del jugador normalmente, si sale un 5 o un 6 el sistema asignará un riesgo al jugador.
RF16	El sistema deberá almacenar el conjunto de riesgos y empleados disponibles.
RF17	Al final de cada turno el sistema descontará al jugador el salario correspondiente a la suma del salario semanal de todos sus empleados.
RF18	El sistema eliminará de la partida, automáticamente, al jugador que se quede sin dinero.
RF19	El sistema deberá, ante cualquier cambio en los atributos del jugador (número de empleados, presupuesto...), actualizar la interfaz.

3. REQUISITOS

- Requisitos funcionales
 - Iteración 6
 - RF04, RF10, RF11, RF13.
 - Iteración 7
 - RF01, RF14.
 - Iteración 9
 - RF02, RF03, RF05, RF07, RF18.
 - Iteración 10
 - Tween, clasificación y estilo gráfico.

#	DESCRIPCIÓN
RF01	El juego deberá ser jugado en modo multijugador.
RF02	El juego permitirá configurar parámetros iniciales como: el número de empleados iniciales o el tablero de entre los disponibles.
RF03	El juego se desarrollará por turnos.
RF04	El juego automatizará el cálculo de valores tales como el SPI, CPI o el coste total de una fase a partir de unos parámetros introducidos por el jugador.
RF05	El juego continuará pese a que un jugador abandone la partida.
RF06	El juego contará con un dado como método para que el jugador avance las casillas correspondientes.
RF07	El juego finalizará cuando un jugador gane, todos se queden sin dinero menos uno o todos, menos uno, abandonen la partida.
RF08	El tablero tendrá una casilla de salida y una de llegada o meta.
RF09	Al finalizar la partida, el sistema mostrará un sumario con la posición de cada jugador en la partida.
RF10	Cuando un jugador llegue a un hito, el sistema mostrará el reporte de actuación de la fase pasada, la ventana de asignación de empleados y el plan de proyecto para la siguiente fase.
RF11	El jugador deberá introducir, en el reporte de actuación de la fase, el valor ganado y el sistema comprobará que el valor es correcto. Después, el sistema mostrará los valores del CPI y el SPI.
RF12	El jugador, utilizando la ventana de asignación de empleados, deberá seleccionar los empleados que desea tener disponibles para la siguiente fase.
RF13	El jugador deberá introducir una nueva planificación para la siguiente fase al inicio y al llegar a un hito.
RF14	El sistema permitirá tener acceso en todo momento a la información relevante del proyecto como el plan de proyecto, el reporte de actuación, los empleados o el dinero disponible.
RF15	Si se lanza el dado y sale un número entre 1 y 4 el sistema avanzará la ficha del jugador normalmente, si sale un 5 o un 6 el sistema asignará un riesgo al jugador.
RF16	El sistema deberá almacenar el conjunto de riesgos y empleados disponibles.
RF17	Al final de cada turno el sistema descontará al jugador el salario correspondiente a la suma del salario semanal de todos sus empleados.
RF18	El sistema eliminará de la partida, automáticamente, al jugador que se quede sin dinero.
RF19	El sistema deberá, ante cualquier cambio en los atributos del jugador (número de empleados, presupuesto...), actualizar la interfaz.

3. REQUISITOS

- Requisitos no funcionales
 - 5 tipos de requisitos (ISO 25010):
 - Portabilidad: adaptabilidad a distintas versiones.
 - Usabilidad: inteligibilidad de la interfaz.
 - Disponibilidad: accesibilidad del código.
 - Compatibilidad: interoperabilidad entre distintos dispositivos.
 - Rendimiento: correcta gestión de los recursos.

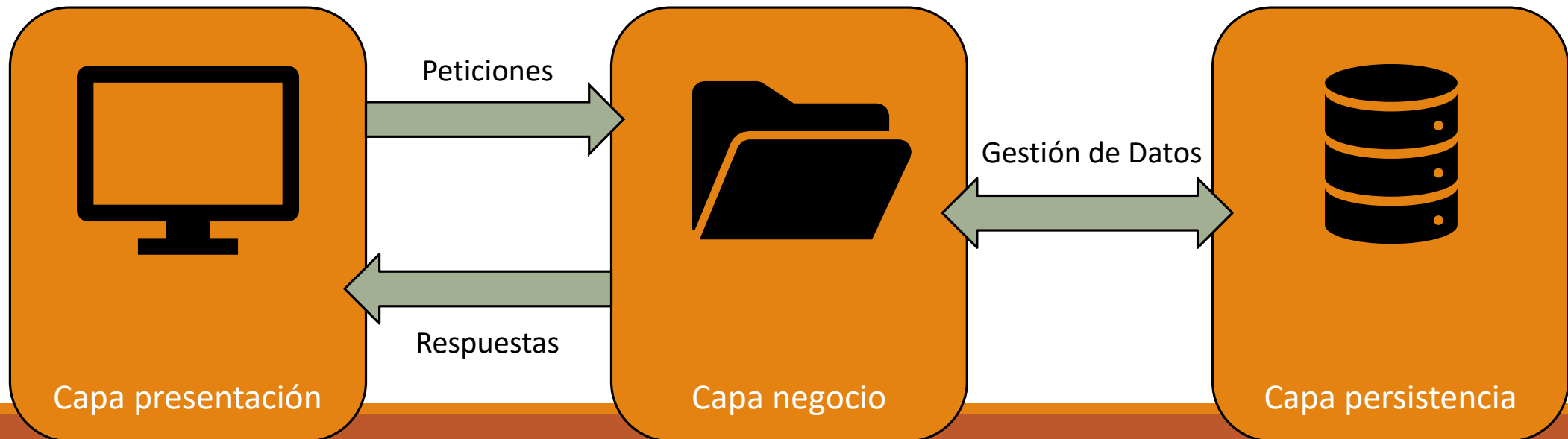
#	TIPO	DESCRIPCIÓN	RELEVANCIA
RNF01	Portabilidad	El juego deberá poder ejecutarse en versión escritorio en Microsoft Windows y en móviles y tabletas Android.	Muy alta
RNF02	Usabilidad	La interfaz del juego deberá ser intuitiva.	Alta
RNF03	Disponibilidad	La información relevante para el juego deberá ser almacenada en ficheros XML.	Muy alta
RNF04	Compatibilidad	Los jugadores que se encuentren en la misma partida deberán tener la posibilidad de jugar desde distintos dispositivos (juego cruzado).	Alta
RNF05	Rendimiento	El juego deberá hacer un uso máximo de memoria de 2GB.	Alta
RNF06	Rendimiento	El juego deberá superar en todo momento los 24 fotogramas por segundo.	Alta

ÍNDICE

- Introducción
- Herramientas y metodología
- Requisitos
- **Diseño e implementación**
- Pruebas
- Demo
- Conclusiones

4. DISEÑO E IMPLEMENTACIÓN

- Arquitectura en 3 capas
 - Capa de presentación: Gestión de las distintas pantallas del juego.
 - Capa de negocio: Gestión de la lógica del juego.
 - Capa de persistencia: Gestión y manipulación de los datos almacenados.

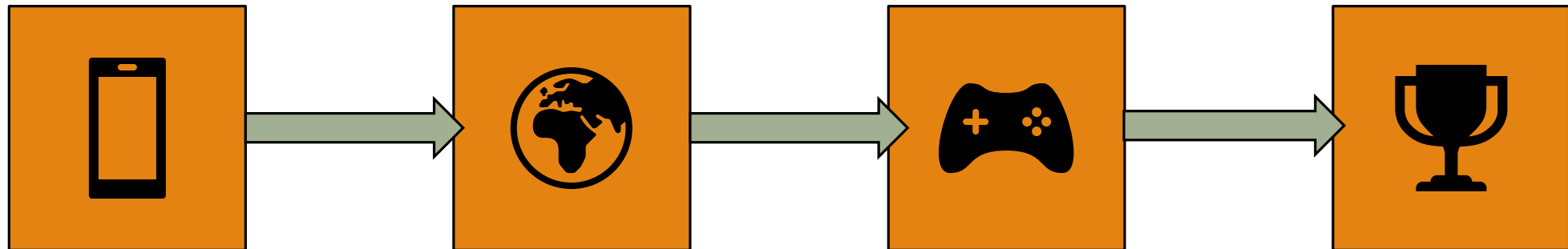


4. DISEÑO E IMPLEMENTACIÓN

- Capa de presentación

- Diseño: Escenas

- MENU: A través de ella se accede al juego y se visualizan las puntuaciones.
 - LOBBY: Gestiona la conexión con el servidor *Photon*.
 - GAME: En ella se desarrolla la partida.
 - STATS: Ofrece información sumariada de la partida finalizada.

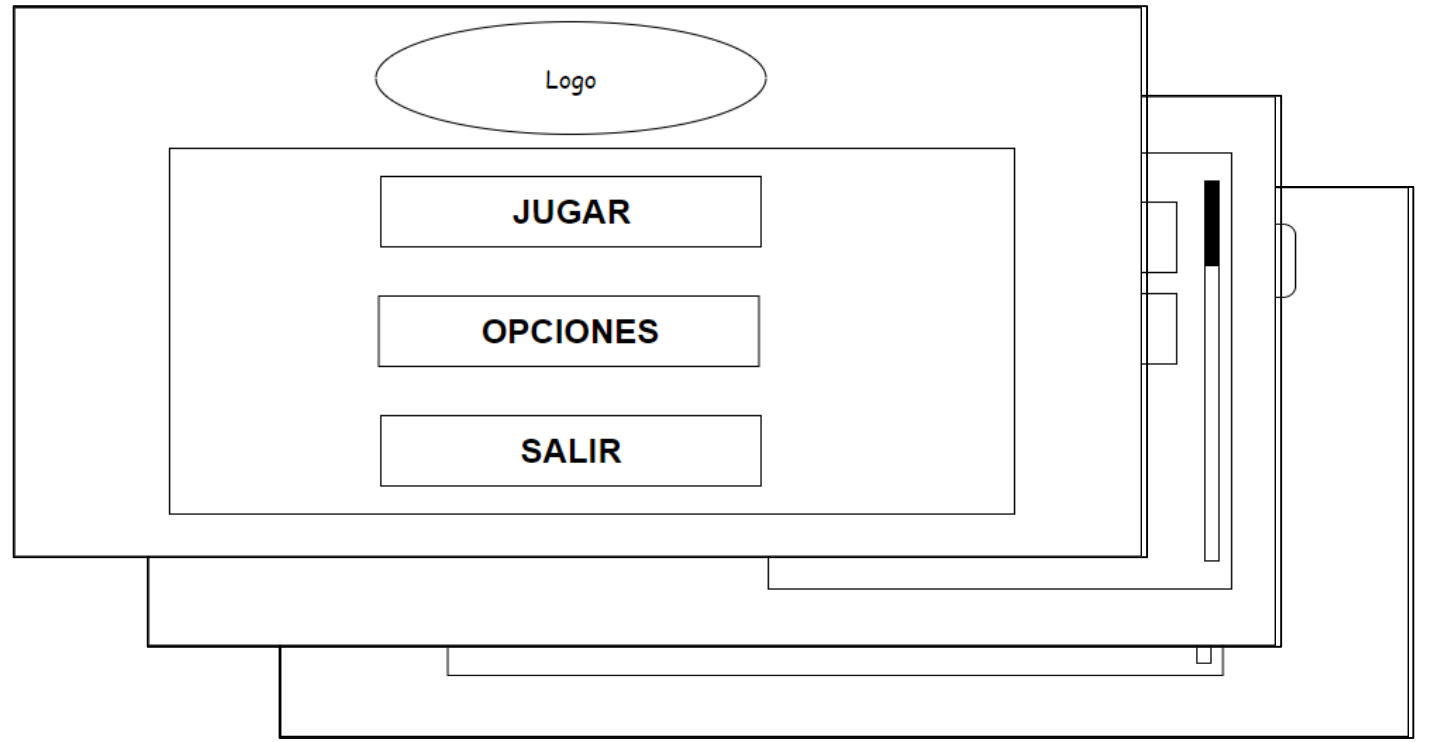


4. DISEÑO E IMPLEMENTACIÓN

- Capa de presentación

- Diseño: *Mockups*

- Maqueta inicial del diseño de la interfaz.
 - Se realizaron 3 mockups:
 - Menú principal.
 - *Lobby* del servidor *online*.
 - Sala de la partida.



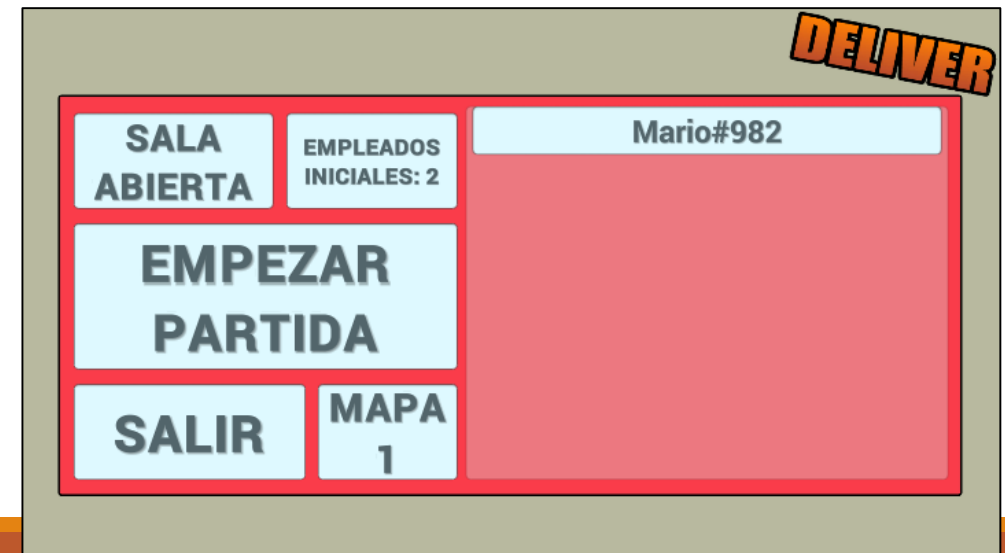
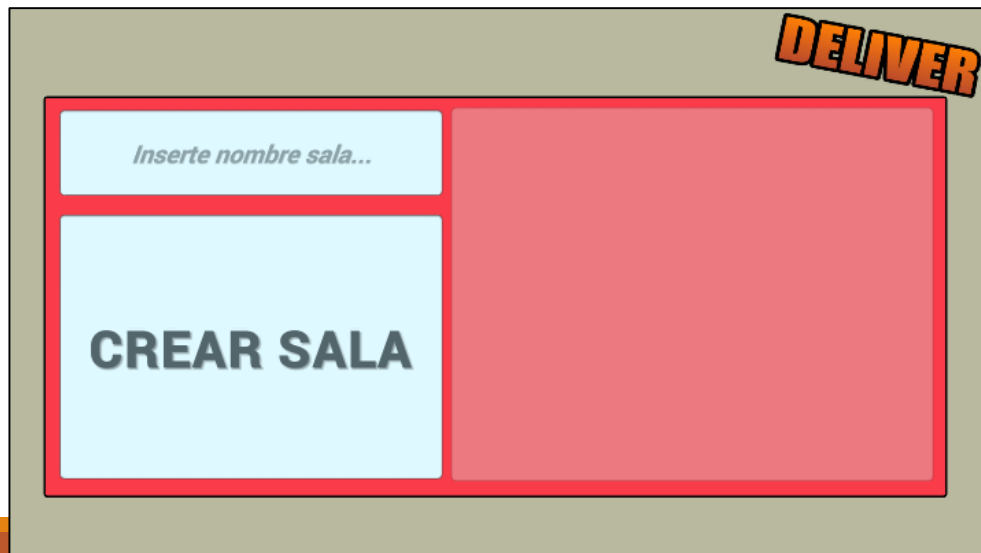
4. DISEÑO E IMPLEMENTACIÓN

- Capa de presentación
 - Implementación: Escena MENU
 - La escena contiene:
 - Elementos interactivos.
 - *Input text* para el nombre.



4. DISEÑO E IMPLEMENTACIÓN

- Capa de presentación
 - Implementación: Escena LOBBY
 - La escena contiene:
 - Dos paneles; uno para crear la sala y otro para gestionarla.
 - Elementos interactivos para navegar ajustar parámetros.







4. DISEÑO E IMPLEMENTACIÓN

- Capa de presentación

- Implementación: Escena GAME
 - La escena contiene:
 - Elementos del juego.
 - Elementos interactivos.
 - Elementos informativos.

RECONFIGURE SU EQUIPO

	Steve Prod: 3 1000\$	<input type="checkbox"/>
	Grady Prod: 3 1000\$	<input checked="" type="checkbox"/>
	Ada Prod: 3 1000\$	<input type="checkbox"/>
	Suzanne Prod: 2 100\$	<input checked="" type="checkbox"/>

CONTINUAR

[illegible]

4. DISEÑO E IMPLEMENTACIÓN

- Capa de presentación
 - Implementación: Escena STATS
 - La escena contiene:
 - Una infografía con la posición de cada jugador al terminar la partida.
 - Un botón para abandonar el juego.



4. DISEÑO E IMPLEMENTACIÓN

- Capa de negocio
 - Diseño: Módulos

JUEGO

TABLERO

JUGADOR

DADO

EMPLEADOS

RIESGOS

FORMULARIOS

GAME MANAGER

MULTIJUGADOR

CONEXIÓN A PHOTON

SALAS

GESTOR DE SALAS

GESTOR MODO
MULTIJUGADOR

MENÚS

MENÚ PRINCIPAL

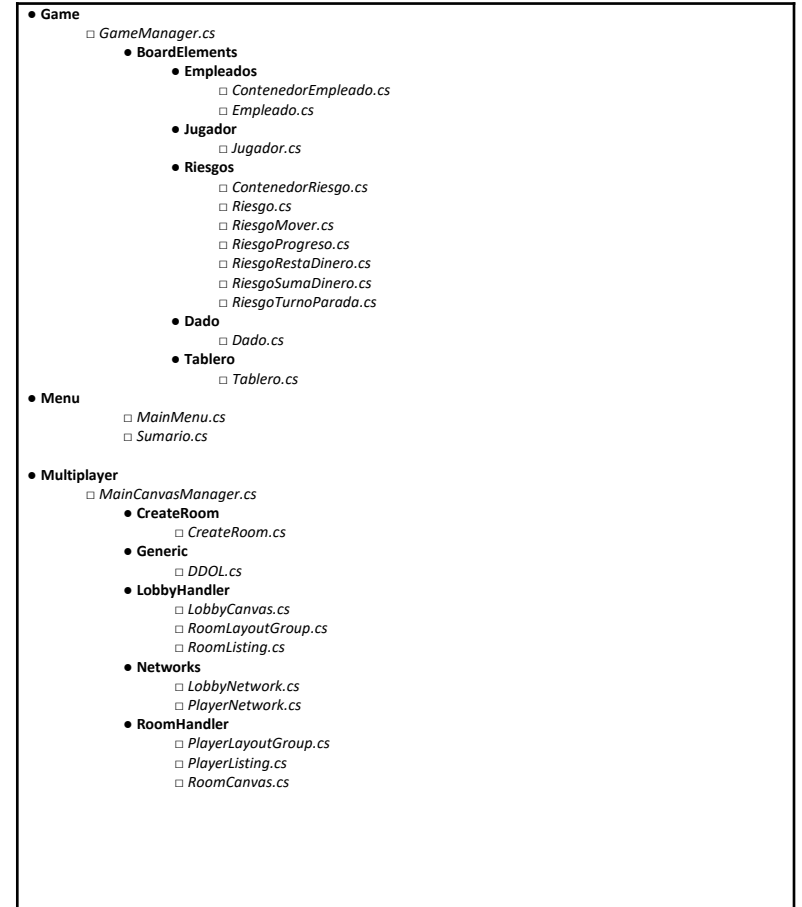
PUNTUACIONES

GESTOR DEL SUMARIO

4. DISEÑO E IMPLEMENTACIÓN

- Capa de negocio

- Implementación: Jerarquía de *scripts*
 - La implementación se hizo en 3 módulos distintos de acuerdo a lo diseñado.
 - Estos módulos se encargan de lo siguiente:
 - Game: Gestión de la lógica del juego durante la partida.
 - Menu: Gestión del menú principal y el sumario.
 - Multiplayer: Gestión del modo *online*.



4. DISEÑO E IMPLEMENTACIÓN

- Capa de negocio
 - Implementación: *Game Manager*
 - Implementación de un patrón de diseño: *Singleton*.

```
60     public static GameManager _instance = null;
61
62     void Start()
63     {
64         // chequeamos si existe una instancia de la clase
65         if (_instance == null) {
66             // si no existe esta sera la empleada a partir de ahora
67             _instance = this;
68         }
69         // si existe y es diferente de esta
70         else if (_instance != this) {
71             // destruimos esta instancia
72             Destroy (gameObject);
73         }
74     }
75
```

4. DISEÑO E IMPLEMENTACIÓN

- Capa de negocio
 - Implementación: *Game Manager*
 - Implementación de un patrón de diseño: *Singleton*.
 - Gestiona el desarrollo de la partida.
 - Funciones:
 - Cargar datos.
 - Gestión de la interfaz.
 - Gestión de los formularios.
 - Gestión del dado.
 - Gestión del cambio de turno.

```
124 public void TirarDado () {
125     int ganador = 0;
126
127     if (PlayerNetwork.instance.jugadores [PlayerNetwork.instance.turno] == jugadorGestionado.GetComponent<PhotonView> ().owner.NickName && permitirTurno) {
128         Dado dScript = GameObject.FindWithTag ("Dado").GetComponent<Dado> ();
129         int puntuacion = dScript.LanzarDado ();
130         Jugador jScript = jugadorGestionado.GetComponent<Jugador> ();
131
132         if (puntuacion >= 5) {
133             Riesgo riesgo = riesgos.riesgos [UnityEngine.Random.Range (0, riesgos.riesgos.Count)];
134             riesgo.EjecutaEfecto(jScript);
135             ganador = jugadorGestionado.GetComponent<Jugador> ().Mover (puntuacion, tablero);
136             if (ganador != 1) { // Si no se ha ganado se muestra el riesgo
137                 cartaRiesgo.SetActive (true);
138                 textRiesgo.GetComponent<Text> ().text = riesgo.nombreRiesgo + "\n" + riesgo.efecto;
139                 permitirTurno = false;
140             }
141         } else {
142             ganador = jugadorGestionado.GetComponent<Jugador> ().Mover (puntuacion, tablero);
143         }
144
145         switch (ganador) {
146             case -1: // JUGADOR ELIMINADO
147                 if (PlayerNetwork.instance.PlayerEliminated ()) {
148                     listaEmpleados.SetActive (false);
149                     datos.SetActive (false);
150                     elementosInteractivos.SetActive (false);
151                     CambiarTurnoJugador ();
152                 } else {
153                     LanzarSumarioPartida ();
154                 }
155                 break;
156             case 0: // TURNO NORMAL
157                 ActualizarInterfaz (false);
158                 CambiarTurnoJugador ();
159                 break;
160             case 1: // GANADOR
161                 ActualizarInterfaz (false);
162                 //PlayerNetwork.instance.ChangeTurn(PhotonNetwork.playerList.Length); // No habra ningun jugador con ID igual al numero de jugadores en la sala.
163                 LanzarSumarioPartida();
164                 break;
```

4. DISEÑO E IMPLEMENTACIÓN

- Capa de negocio

- Implementación: Servidor *Photon*
 - Lo componen un conjunto de clases que gestionan el modo multijugador.
 - Aspectos relevantes:
 - Conexión al servidor.

```
3 public class LobbyNetwork : MonoBehaviour {
4
5     private void Start () {
6         Debug.Log("Start Lobby Network");
7         PhotonNetwork.ConnectUsingSettings ("1.0");
8     }
9
10    private void OnConnectedToMaster() {
11        Debug.Log ("Connecting to master");
12        PhotonNetwork.automaticallySyncScene = true;
13        PhotonNetwork.playerName = PlayerNetwork.instance.pName;
14        PhotonNetwork.JoinLobby (TypedLobby.Default);
15    }
16
17    private void OnJoinedLobby(){
18        Debug.Log ("Joined lobby");
19        if (!PhotonNetwork.inRoom) {
20            MainCanvasManager.instance.LobbyCanvas.transform.SetAsLastSibling ();
21        }
22    }
23 }
24
```

4. DISEÑO E IMPLEMENTACIÓN

- Capa de negocio

- Implementación: Servidor *Photon*
 - Lo componen un conjunto de clases que gestionan el modo multijugador.
 - Aspectos relevantes:
 - Conexión al servidor.
 - PlayerNetwork y métodos PunRPC.

```
106     public void ChangeTurn(int turno){
107         PhotonView.RPC ("RPC_ChangeTurn", PhotonTargets.All, turno);
108     }
109
110     [PunRPC]
111     private void RPC_ChangeTurn(int turno){
112         this.turno = turno;
113         GameManager._instance.CambiarTextoTurno ();
114     }
115
```

4. DISEÑO E IMPLEMENTACIÓN

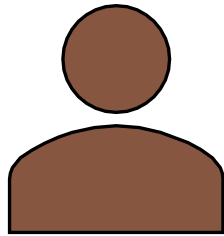
- Capa de negocio

- Implementación: Servidor *Photon*
 - Lo componen un conjunto de clases que gestionan el modo multijugador.
 - Aspectos relevantes:
 - Conexión al servidor.
 - PlayerNetwork y métodos PunRPC.
 - DDOL.

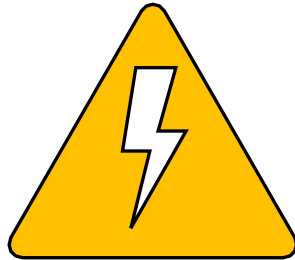
```
3 public class DDOL : MonoBehaviour {  
4  
5     private void Awake () {  
6         DontDestroyOnLoad (this); // Se usa para que no se destruya el objeto que gestiona el multiplayer  
7         // entre escenas  
8     }  
9 }  
10
```

4. DISEÑO E IMPLEMENTACIÓN

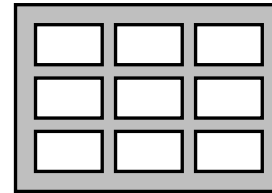
- Capa de datos
 - Diseño: Elementos almacenables



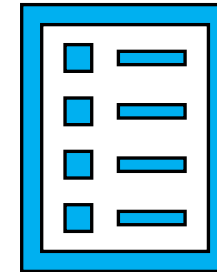
Empleados



Riesgos



Tableros



Clasificación

4. DISEÑO E IMPLEMENTACIÓN

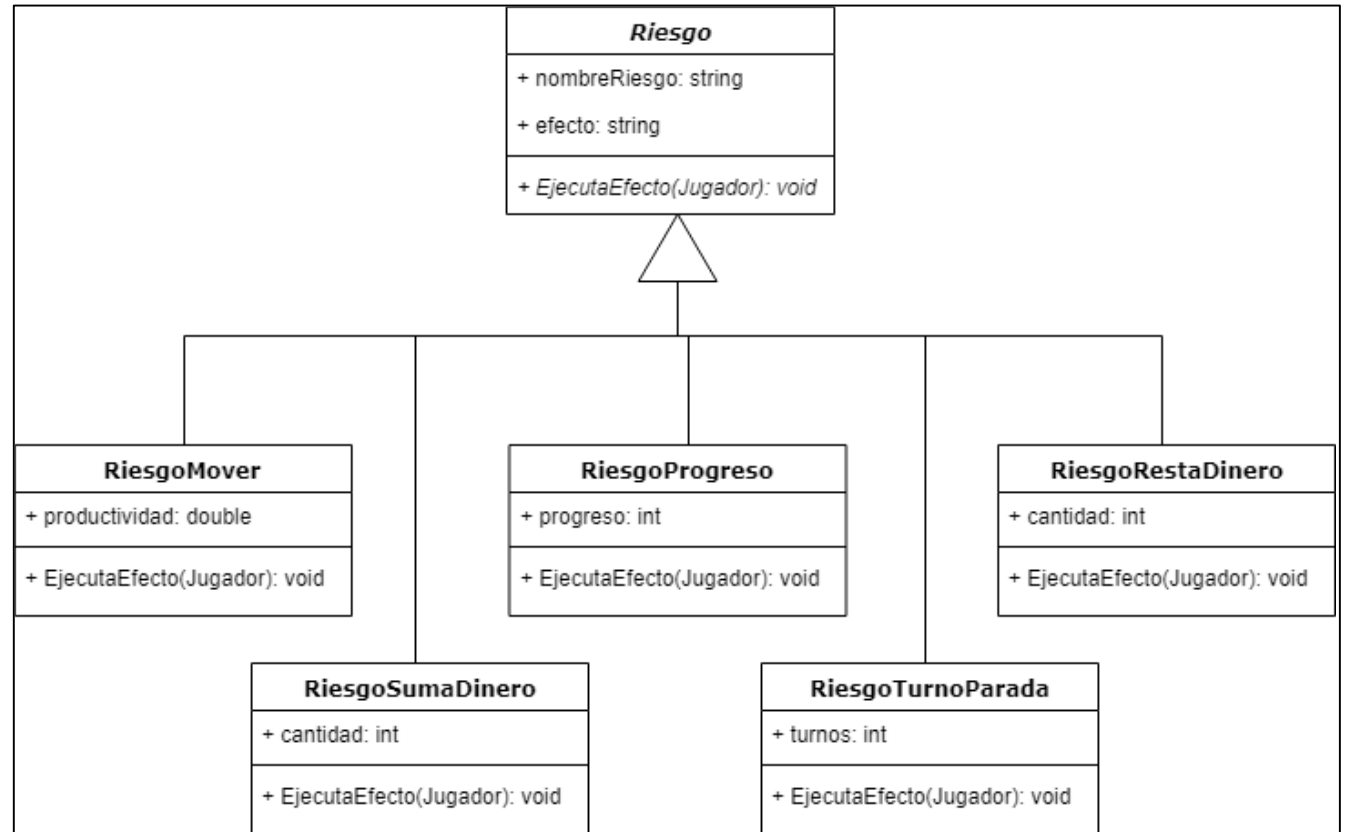
- Capa de datos
 - Implementación: Empleados
 - Documento XML.
 - Clase Parser.
 - Clase básica: Empleado.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ColeccionEmpleado>
3   <Empleados>
4     <Empleado Nombre="Bill">
5       <Productividad>3</Productividad>
6       <Salario>1000</Salario>
7     </Empleado>
8     <Empleado Nombre="Steve">
9       <Productividad>3</Productividad>
10      <Salario>1000</Salario>
11    </Empleado>
12  </Empleados>
13 </ColeccionEmpleado>
```

```
7 [XmlRoot("ColeccionEmpleado")]
8 public class ContenedorEmpleado {
9
10   [XmlArray("Empleados")]
11   [XmlArrayItem("Empleado")]
12   public List<Empleado> empleados = new List<Empleado>();
13
14   public static string xml_path = "xmls/empleados";
15
16   public static ContenedorEmpleado CargarEmpleados(){
17     TextAsset _xml = Resources.Load<TextAsset> (xml_path);
18
19     XmlSerializer serializer = new XmlSerializer (typeof(ContenedorEmpleado));
20
21     StringReader reader = new StringReader (_xml.text);
22
23     ContenedorEmpleado _contenedor = serializer.Deserialize (reader) as ContenedorEmpleado;
24
25     reader.Close();
26
27     return _contenedor;
28   }
29 }
```


4. DISEÑO E IMPLEMENTACIÓN

- Capa de datos
 - Implementación: Riesgos
 - Documento XML.
 - Clase Parser.
 - Clase básica: Riesgo.
 - Herencia de riesgo.



4. DISEÑO E IMPLEMENTACIÓN

- Capa de datos

- Implementación: Tableros
 - CSV estructurado. Formato 8x40.
 - Primera línea para el presupuesto inicial.
 - Cada número representa un tipo de casilla distinta.

[illegible]

4. DISEÑO E IMPLEMENTACIÓN

- Capa de datos

- Implementación: Clasificación
 - Servicio externo: *dreamlo*.
 - Uso de la librería WWW de C#.
 - Llamadas al servicio a través de enlaces web.

Here is your **private** url. Copy and paste this somewhere.

Do not tell anyone about this link.

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

WebGL builds hosted at itch.io and other services may need SSL to work!

Want to use **https (SSL)**? [Donate \\$5 or more](#) and let me know.

Want to store **more than 1000 scores**? [Contact me](#).

(If you have a limit of 1000 scores and another score comes in, the lowest will get bumped out.)

You copy and pasted that somewhere and are never going to tell anyone right?

You can not have an asterisk * character in your URL, scores, usernames, etc.

Your Leaderboard

Highest 1000 scores.

Name	Score
mario	30 delete
player2	30 delete
player1	30 delete
carlos	15 delete
mario2	15 delete
aaaaaaa	15 delete
Ludi	15 delete
JUGADOR1	15 delete
David+el+Nohomo	15 delete
pebe	15 delete
LuDisan	15 delete
osle	15 delete
Mario2	15 delete
Mario	10 delete
eeey	10 delete

[Remove All Scores](#)

Adding and deleting scores

Changes and updates to your leaderboard are made through simple [http get requests](#) using your **private** url.

A player named **Carmine** got a score of **100**. If the same name is added twice, we use the higher score.

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

A player named **Carmine** got a score of **1000 in 90 seconds**.

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

A player named **Carmine** got a score of **1000 in 90 seconds** and is **Awesome**.

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

Delete **Carmine's** score

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

Clear **all** scores

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

Save a trip to the server by combining "add" with returning data: "add-pipe", "add-xml" or "add-quote"

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e>

Codes for Unity Example

Here are just your public and private code so that you can cut and paste them into the sample code for Unity.

Private Code (It's long, get all of it!)

[5c4f1446b6397e0c24c5976e](#)

Public Code

[5c4f1446b6397e0c24c5976e](#)

Getting your scores

Reading of data is performed by using your **public** url.

Get your data as **XML**:

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e/xml>

Get your data as **json**:

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e/json>

Get your data as **pipe delimited**:

<http://dreamlo.com/lb/5c4f1446b6397e0c24c5976e/pipe>

ÍNDICE

- Introducción
- Herramientas y metodología
- Requisitos
- Diseño e implementación
- **Pruebas**
- Demo
- Conclusiones

5. PRUEBAS

- Pruebas unitarias y de integración

- Problemática de las pruebas unitarias.
- Pruebas realizadas:
 - Funciones de *log*.
 - Ejecuciones del juego.
 - Test con *Unity Test Runner (NUnit)*

```
1 using UnityEngine;
2 using UnityEditor;
3 using UnityEngine.TestTools;
4 using NUnit.Framework;
5 using System.Collections;
6
7 public class MainMenuTest {
8
9     [UnityTest]
10    public IEnumerator TestLobbyJugar() {
11        GameObject objeto = new GameObject();
12        objeto.AddComponent<MainMenu> ();
13
14        Assert.IsNull (objeto.GetComponent<MainMenu>().nombre);
15
16        objeto.GetComponent<MainMenu> ().LobbyJugar ();
17
18        Assert.IsNotNull (objeto.GetComponent<MainMenu> ().nombre);
19        yield return null;
20    }
21 }
```

5. PRUEBAS

- Pruebas de sistema

- Pruebas de portabilidad

- Comprobación de que el juego pudiera ejecutarse multiplataforma.
 - De acuerdo a los requisitos no funcionales las plataformas escogidas fueron:
 - Ordenador con sistema operativo Windows.
 - Dispositivos móviles con sistema operativo Android.

- Pruebas de compatibilidad

- Consistía en probar que el juego pudiera ser jugado de manera cruzada.
 - Se comprobó el correcto desarrollo de una partida entre el ordenador y el móvil.

5. PRUEBAS

- Pruebas de sistema
 - Pruebas de rendimiento
 - Se buscaba que el juego no usara más de 2GB de memoria y no bajara de los 24 fotogramas por segundo.
 - Unity *Profiler*.



5. PRUEBAS

- Pruebas de sistema
 - Pruebas de rendimiento
 - Uso de memoria.
 - Uso = 0,95 GB. Test superado.

```
Simple ▾
Used Total: 245.7 MB  Unity: 135.1 MB  Mono: 18.8 MB  GfxDriver: 96.9 MB  FMOD: 1.2 MB  Video: 0 B  Profiler: 13.7 MB
Reserved Total: 390.0 MB  Unity: 273.2 MB  Mono: 48.7 MB  GfxDriver: 96.9 MB  FMOD: 1.2 MB  Video: 0 B  Profiler: 20.0 MB
Total System Memory Usage: 0.95 GB

Textures: 2113 / 168.2 MB
Meshes: 46 / 451.0 KB
Materials: 41 / 55.0 KB
AnimationClips: 0 / 0 B
AudioClips: 0 / 0 B
Assets: 3178
GameObjects in Scene: 302
Total Objects in Scene: 1202
Total Object Count: 4380
GC Allocations per Frame: 160 / 6.0 KB
```


5. PRUEBAS

- Pruebas de sistema

- Pruebas de rendimiento

- Uso de CPU.

- Tasa de refresco de la CPU: 16,16 ms.

- $FPS = \frac{1}{T} = 61$ fotogramas por segundo. Test superado.

Hierarchy		CPU:16.16ms GPU:2.80ms					Q	
Overview		Total	Self	Calls	GC Alloc	Time ms		
▶	Initialization.PlayerUpdateTime	59.0%	0.1%	1	0 B	9.54		
	EditorOverhead	18.6%	18.6%	2	0 B	3.01		
▶	Profiler.CollectGlobalStats	9.6%	0.6%	1	0 B	1.55		
▶	Camera.Render	5.6%	0.4%	1	0 B	0.91		
▼	Update.ScriptRunBehaviourUpdate	1.0%	0.0%	1	0 B	0.17		
▶	BehaviourUpdate	1.0%	0.0%	1	0 B	0.17		
▶	PreLateUpdate.ScriptRunBehaviourLateUpdate	0.7%	0.0%	1	0 B	0.12		
▶	PostLateUpdate.PlayerUpdateCanvases	0.6%	0.0%	1	0 B	0.11		
▶	PostLateUpdate.UpdateAudio	0.4%	0.0%	1	0 B	0.07		
	Profiler.CollectUIStats	0.4%	0.4%	1	0 B	0.06		
	UGUI.Rendering.EmitWorldScreenspaceCameraGeometry	0.3%	0.3%	1	0 B	0.05		
▶	EarlyUpdate.UpdateMainGameViewRect	0.3%	0.0%	1	0 B	0.04		

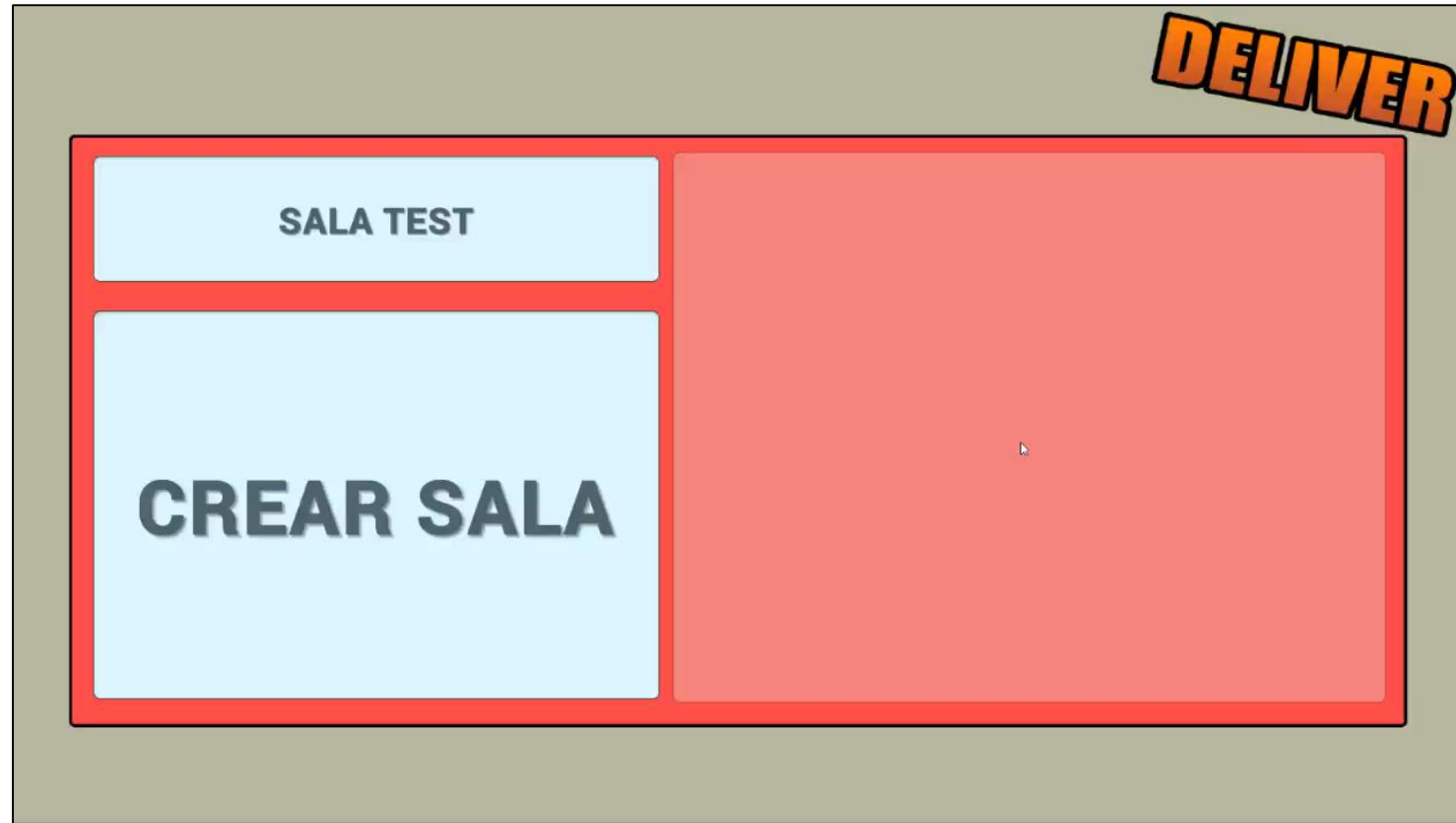
5. PRUEBAS

- Pruebas de aceptación
 - Consistieron en la validación por parte un conjunto de usuarios.
 - Los usuarios que validaron el juego fueron:
 - Director del proyecto.
 - Una empresa de desarrollo de videojuegos llamada *Concano Games*.
 - Un conjunto de usuarios que ya habían jugado a la versión física.
 - En el futuro será probado por los alumnos de gestión de proyectos.

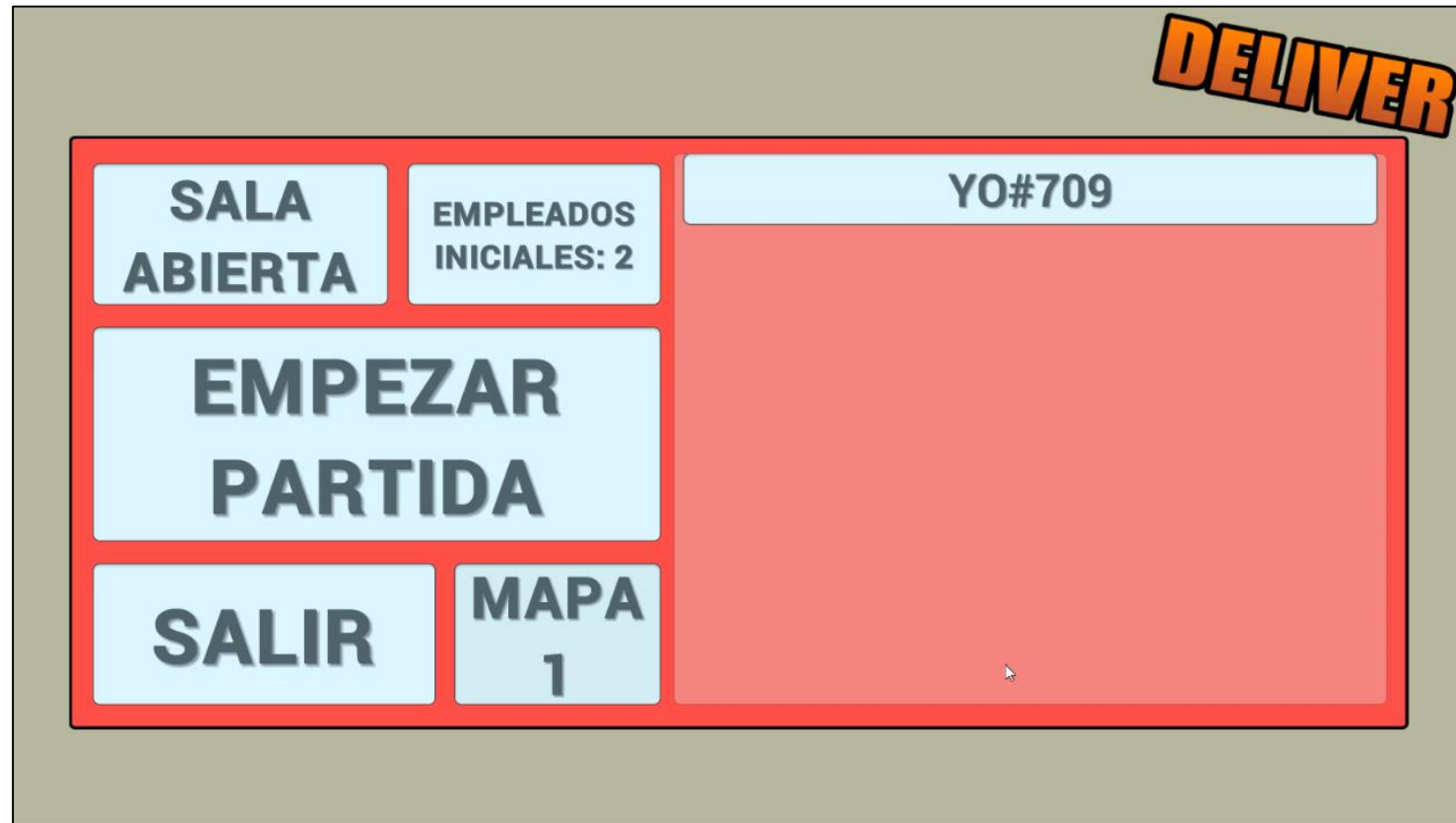
ÍNDICE

- Introducción
- Herramientas y metodología
- Requisitos
- Diseño e implementación
- Pruebas
- **Demo**
- Conclusiones

6. DEMO



6. DEMO



6. DEMO



ÍNDICE

- Introducción
- Herramientas y metodología
- Requisitos
- Diseño e implementación
- Pruebas
- Demo
- **Conclusiones**

7. CONCLUSIONES

- Conclusiones

- El juego cumple con el objetivo principal que se había establecido.
- Además añade nuevas funcionalidades respecto al formato físico:
 - Variabilidad de tableros.
 - Recursos aleatorios:
 - Empleados
 - Riesgos
 - Multiplataforma (escritorio y móvil).
 - Modo multijugador.
 - Juego cruzado.

7. CONCLUSIONES

- Trabajos futuros
 - Funcionalidad nuevas que se espera implementar:
 - Internacionalización del juego.
 - Inclusión de sonidos.
 - Pielés temáticas.
 - BD de usuarios.